Navy Personnel Research and Development Center
San Diego, California 92152-6800 TN-91-20 Volume 1 August 1991

(2)

Human-Computer Interfaces for Tactical Decision Making, Analysis, and Assessment Using Artificially Intelligent Platforms:

Volume 1, Software Design and Database Descriptions for BATMAN & ROBIN

Pat-Anthony Federico
Randy R. Ullrich
Brian L. Van de Wetering
Christel I. Tomlinson
Dean J. E. Long
Fred R. E. Long
Thomas E. Bridges

NPRDC-TN-91-20 August 1991

Human-Computer Interfaces for Tactical Decision Making, Analysis, and Assessment Using Artificially Intelligent Platforms: Volume 1, Software Design and Database Descriptions for BATMAN & ROBIN

Pat-Anthony Federico

Navy Personnel Research and Development Center

Randy R. Ullrich
Brian L. Van de Wetering
Christel I. Tomlinson
Dean J. E. Long
Fred R. E. Long
Thomas E. Bridges

Systems Engineering Associates

Reviewed and released by
William E. Montague
Director, Training Technology Department (Acting)

Navy Personnel Research and Development Center
San Diego, California 92152-6800

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	August 1991	Final--Apr 89-Aug 91

4. TITLE AND SUBTITLE
Human-Computer Interfaces for Tactical Decision Making, Analysis, and Assessment Using Artificially Intelligent Platforms: Volume 1, Software Design and Database Descriptions for BATMAN & ROBIN

5. FUNDING NUMBERS
PE 0602763N, WU 522-801-013-03.04;
PE 0603720N, PR Z-1772, TA ET08;
PE 0205604N, TA X1977

6. AUTHOR(S)
Pat-Anthony Federico, Randy R. Ullrich, Brian L. Van de Wetering, Christel I. Tomlinson, Dean J. E. Long, Fred R. E. Long, and Thomas E. Bridges

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Navy Personnel Research and Development Center
San Diego, California 92152-6800

8. PERFORMING ORGANIZATION REPORT NUMBER
NPRDC-TN-91-20

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of the Chief of Naval Research (OCNR-222)
Chief of Naval Operations (PERS-11)
Space and Naval Warfare Systems Command (SPAWAR 159-4)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This technical note contains the introduction, artificial intelligence techniques, software design, and database descriptions for the Battle-Management Assessment System and Raid Originator Bogie Ingress (BATMAN & ROBIN). These software systems are being developed to assess how well individuals can allocate, deploy, and manage air, surface, and/or subsurface tactical assets during simulated sea battles in many warfare areas. Together they form a desk-top, computer-based, performance-measurement system incorporating high resolution graphics, low level modelling, artificial intelligence techniques, to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators. Two of the major contributions of these dual systems are very friendly human-computer interfaces and automated performance assessment. Because of the nature of their generic software and independent databases, as well as the potential for incorporating different computer models, BATMAN & ROBIN can be used for a variety of functions: (a) training and testing tactical knowledge, (b) planning and decision aiding for tactical situations, (c) developing and evaluating tactics themselves, (d) analyzing and evaluating various tactical sensor, weapon, and communication systems, (e) frontending sophisticated tactical computer models and complex databases, (f) interfacing tactical artificial intelligent and expert systems, (g) generating rapidly scenarios for tactical trainers, (h) prototyping complicated scenarios for major wargaming systems, (i) orienting novices to facets of naval warfare, (j) evaluating tactical display symbologies and formats, and (k) providing an experimental environment for studying tactical decision making.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Human-Computer Interface; Direct-Manipulation Interfaces; Artificial Intelligence; Tactical Decision Making, Analysis, and Assessment; Tactical Development and Evaluation; Wargaming; Computer Simulation and Modeling; Decision Support System	145
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

FOREWORD

The funding provided by the Office of Naval Technology (Support Technology Directorate), Program Element 0602763N, Work Unit 522-801-013-03.04; Deputy Chief of Naval Operations for Manpower, Personnel, and Training (Total Force Training and Education), Program Element 0603720N, Project Z-1772, Task ET08; and Space and Naval Warfare Systems Command (Advanced Tactical Data Link Systems), Program Element 0205604N, Task X1977; for this documented development is appreciated and acknowledged.

The general goal of this effort was to develop friendly human-computer interfaces for performance assessment and rapid scenario generation for the Navy's tactical training communities as well as research and development centers to frontend different computer models and databases, when necessary, for a variety of functions, which are identified in the introduction of this document.

The assistance through the years of a number of former and present University of California, San Diego students, who were brought on board under the San Diego State University Foundation contract to support the Navy laboratories in this area, in developing and evaluating several versions of our software is appreciated and acknowledged. These include Debbie Bartolome, Steve Bickel, Chris Cassella, Phil Cohen, Bill Kamm, Di Lacerna, Bill Limm, Nina Liggett, Glen Little, Jerry Lugert, Rob McCarter, Tony Meadors, Lorna Mildice, Sole Millonida, Dan Nadir, Alex Olender, Regina Peck, Jeff Roorda, Chris Ryan, Karl Schricker, Ellen Schuller, Alice Shimada, and Brian Smithey.

The support of the staff of Commander, Naval Air Force, United States Pacific Fleet; Commander, Fighter Airborne Early Warning Wing Pacific; and Commander, Patrol Wings Pacific is appreciated and acknowledged. Also, the support and assistance of the Commanding Officers, Executive Officers, and instructors at the operational testbed sites for our software is appreciated and acknowledged, specifically: Carrier Airborne Early Warning Weapons School, NAS Miramar; Fighter Squadron 124, NAS Miramar; Tactical Training Team, NAS Moffett Field; Sea-Based Weapons and Advanced Tactics Squadron, NAS North Island; Marine Aviation Weapons and Tactics Squadron One, MCAS Yuma; Officer Tactical Training Department, Fleet Combat Training Center Pacific; and Wargaming Department, Tactical Training Group Pacific.

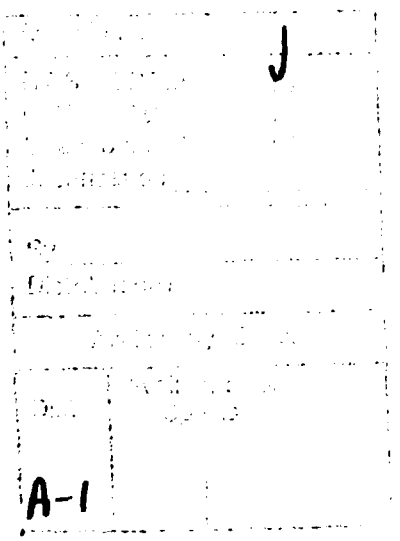
The encouragement, and foresight to use our interfaces, of a large number of individuals at many research and development centers, which are transition sites for our software, specifically: Naval Air Development Center, Naval Surface Warfare Center, Naval Research Laboratory, Naval Training Systems Center, Naval Ocean Systems Center, Naval Weapons Center, Naval Warfare Analysis Center, Applied Physics Laboratory Johns Hopkins University, and Canadian Defense and Civil Institute of Environmental Medicine, as well as the Naval Postgraduate School and Naval War College, is appreciated and acknowledged. Our software is contributing to many research and development projects as well as warfare analysis and wargaming at these sites.

Special thanks to LT Fred Buoni, formerly of the Naval Postgraduate School, for programming the ASW detection models which we have incorporated into our

software; LCDR Howard Peck, formerly of this Command, and Bob MacDougall, Pacer Systems, Inc., for contributing to our tactical knowledge engineering efforts; Bill Root, graduate student, Department of Mathematics, University of California, San Diego, for producing an algorithm which optimizes detections for simulated platforms; Dr. Dan Sunday, APL, for providing the digitized mapping software; Randy Ullrich, Brian Van de Wetering, Christel Tomlinson, Dean Long, Fred Long, and Tom Bridges, SEA, Inc., for their programming expertise; and Jim Bolwerk, Training Officer, AIR-PAC, for his continuous support and assistance through the years.

Also, special thanks to CAPT Bart Bacon, former Commanding Officer, Dr. Jim McMichael, former Technical Director, and Dr. Ed Aiken, former Department Director, all of this command, for their support.

William E. Montague
Director, Training Technology (Acting)



Summary

This technical note contains the introduction, artificial intelligence techniques, software design, and database descriptions for the Battle-Management Assessment System and Raid Originator Bogie Ingress (BATMAN & ROBIN). Other documentation in preparation will discuss the human-computer interfaces employed in BATMAN & ROBIN.

Part I: Introduction

BATMAN is being developed to assess how well individuals can allocate, deploy, and manage air, surface, and/or subsurface tactical assets during simulated sea battles in many warfare areas. ROBIN is being developed to generate rapidly Red force raids comprised of a large number of air, surface, and/or subsurface tactical assets against Blue naval task forces or land bases in many warfare theaters. In order to complete the creation of a scenario, the user also specifies in ROBIN Blue force tactical resources that will be available in BATMAN for allocation, deployment, and management as well as Green or neutral force air, surface, and/or subsurface movements. ROBIN creates scenarios that can be saved, and subsequently presented sequentially or randomly in BATMAN. Together BATMAN & ROBIN form a desk-top, computer-based, performance-measurement system incorporating high resolution graphics, low level modeling, and artificial intelligence techniques to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators. Two of the major contributions of these dual systems are very friendly human-computer interfaces and automated performance measurement.

Because of the nature of their generic software and independent databases, as well as the potential for incorporating different computer models, BATMAN & ROBIN can be used for a variety of functions: (a) training and testing tactical knowledge, (b) planning and decision aiding for tactical situations, (c) developing and evaluating tactics themselves, (d) analyzing and evaluating various tactical sensor, weapon, and communication systems, (e) frontending sophisticated tactical computer models and complex databases, (f) interfacing tactical artificial intelligent and expert systems, (g) generating rapidly scenarios for tactical trainers, (h) prototyping complicated scenarios for major wargaming systems, (i) orienting novices to facets of naval warfare, (j) evaluating tactical display symbologies and formats, and (k) providing an experimental environment for studying tactical decision making.

Part II: Artificial Intelligence

The second part of the documentation defines BATMAN & ROBIN's artificial intelligence or smart platform behavior, including knowledge-based finite state automata and associated state-transition rules for specific missions.

Part III: Software Design

The third part of the documentation deals with BATMAN & ROBIN's software design, including descriptions of the software data structures, packages, and interfaces.

Part IV: Database Descriptions

The fourth part of the documentation describes BATMAN & ROBIN's database descriptions, including parameter, scenario, graphic, and user databases.

Contents

Part I: Introduction

Part II: Artificial Intelligence

Part III: Software Design

Part IV: Database Descriptions

List of Tables

List of Figures

Part I: Introduction

1.0 BATMAN & ROBIN

Battle-Management Assessment System (BATMAN) is being developed to assess how well individuals can allocate, deploy, and manage air, surface, and/or subsurface tactical assets during simulated sea battles in many warfare areas. Raid Originator Bogie Ingress (ROBIN) is being developed to generate rapidly Red force raids comprised of a large number of air, surface, and/or subsurface tactical assets against Blue naval task forces or land bases in many warfare theaters. In order to complete the creation of a scenario, the user also specifies in ROBIN Blue force tactical resources that will be available in BATMAN for allocation, deployment, and management as well as Green or neutral force air, surface, and/or subsurface movements. ROBIN creates scenarios that can be saved, and subsequently presented sequentially or randomly in BATMAN. Together BATMAN & ROBIN form a desk-top, computer-based, performance-measurement system incorporating high resolution graphics, low level modeling, and artificial intelligence techniques to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators. Two of the major contributions of these dual systems are very friendly human-computer interfaces and automated performance measurement.

Since they present an animated simulation, model, metaphor, or microworld to the user, BATMAN & ROBIN employ direct-manipulation, human-computer interfaces (Hutchins, Hollan, & Norman, 1986; Shneiderman, 1982) where graphic objects, e.g., aircraft or ship silhouettes, are continuously depicted, moved, and queried by the operator physically moving and clicking a mouse resulting in immediately visible impact on the icons. These systems assume that the user has some knowledge of Blue, Red, and Green force platforms, sensors, weapons, and tactics. BATMAN & ROBIN use databases which are independent of the simulation software to store the parameters, attributes, and characteristics of Blue, Red, and Green platforms. Currently, these values are unclassified or sanitized; however, they can be made classified by using the friendly graphic interface. BATMAN assesses the tactical decision making of the individual managing the entire battle, or any of its components in terms of composite warfare structure, by measuring performance automatically and objectively against multivariate criteria which are immediately fedback to the user at the end of each scenario. These measures are saved by the system for subsequent statistical analyses, and are available for formative and summative evaluations of performance.

BATMAN & ROBIN are written in the "C" programming language (Kernighan & Ritchie, 1988) and currently run on the Sun-4 family of computers, e.g., 110, 260C, 280S, Sparcstation 1, 2, 330, and 370 as well as the Navy's Desk-Top Tactical Computer (DTC) 2 under Sun Microsystems' Release 4.1.1 of the UNIX operating system. These systems are completely documented and properly commented to facilitate integration of various validated and verified computer models and databases to these friendly human-computer interfaces. The generic nature of BATMAN & ROBIN allows the user to add or delete platforms at will without rewriting the software. Also, the modularity of the code permits the incorporation of different computer models for various sensor, weapon, communication, and environmental systems. The Sun-4 family of computers allows the simultaneous running of models written in different languages, e.g., "C", ADA, MODULA-2, FORTRAN 77, PASCAL, and COMMON LISP.

Because of the nature of their generic software and independent databases, as well as the potential for incorporating different computer models, BATMAN & ROBIN can be used for a variety of functions: (a) training and testing tactical knowledge, (b) planning and decision aiding for tactical situations, (c) developing and evaluating tactics themselves, (d) analyzing and evaluating various tactical sensor, weapon, and communication systems, (e) frontending sophisticated tactical computer models and complex databases, (f) interfacing tactical artificial intelligent and expert systems, (g) generating rapidly scenarios for tactical trainers, (h) prototyping complicated scenarios for major wargaming systems, (i) orienting novices to facets of naval warfare, (j) evaluating tactical display symbologies and formats, and (k) providing an experimental environment for studying tactical decision making. MANY OF THE ABOVE USES ASSUME THE INTEGRATION OF VALID MODELS AND VERIFIED DATABASES INTO BATMAN & ROBIN.

BATMAN & ROBIN are still under development. The present software release is at testbed sites for demonstration, evaluation, and feedback purposes only. AT THIS TIME, BATMAN & ROBIN ARE NOT TO BE USED FOR TACTICAL TRAINING OR DECISION AIDING. THE DATABASES EMPLOYED ARE SANITIZED OR UNCLASSIFIED; PLATFORM PARAMETERS AND COMPUTER MODELS ARE ONLY APPROXIMATE. THAT IS, THE COMPUTER MODELS AND DATABASES HAVE NOT BEEN VALIDATED OR VERIFIED. Our development has concentrated on producing friendly human-computer interfaces for air, surface, subsurface, and electronic warfare and platform-parameter databases as well as making simulated platforms act in an artificially intelligent or smart manner according to defense warnings and weapons status. Validated or verified computer models and databases are available to the interested user throughout the Department of Defense, and can be readily incorporated into BATMAN & ROBIN for a variety of uses as indicated above.

This latest documentation describes the present software design, database descriptions, and artificial intelligence aspects of BATMAN & ROBIN, Version 4.0. It updates earlier documentation which described the background, rationale, software design, and database descriptions for a previous version of BATMAN & ROBIN (Federico, Bickel, Ullrich, Bridges, & Van de Wetering, 1989). It is intended for software engineers familiar with Unix (SunOS Reference Manual, 1990), SunView (SunView Programmer's and System Programmer's Guides, 1990), and the C programming language (Kernighan & Ritchie, 1988). Part II of this document covers BATMAN & ROBIN's artificial intelligence or smart platform behavior, including knowledge-based finite state automata and associated state-transition rules for specific missions. Part III covers BATMAN & ROBIN's software design, including descriptions of the software data structures, packages, and interfaces. Part IV covers BATMAN & ROBIN's database descriptions, including the parameter, scenario, graphic, and user databases. Other documentation in preparation will discuss the human-computer interfaces employed in BATMAN & ROBIN.

2.0 Notational Conventions

The following notational conventions are used throughout this document:

Convention	Meaning
Bold	UNIX filenames, C package names, Object- Definition Database parameters, and document section titles are set in

	bold type to distinguish them from ordinary text. In this documentation, "package" is used to refer to a collection of related C functions and data types grouped in one or more files.
Italics	Italics are used for the names of C functions and variables. In addition, italics are occasionally used to emphasize particular words in the document.
ITALIC CAPITALS	Italic capital letters are used for the names of C data structures.

Part II: Artificial Intelligence

3.0 Artificially Intelligent or Smart Platform Behavior

The objective of smart platforms is to make simulated aircraft, ships, and submarines act artificially intelligently during BATMAN. Blue force behaves more realistically according to its defense warnings, whether or not Red-force hostile action is perceived to be improbable, probable, or imminent, i.e., White, Yellow, or Red, respectively; weapons status, can launch or not launch weapons, i.e., weapons "free" or "tight", respectively; and assigned missions, e.g. fighter, attack, or surveillance aircraft. Many decisions are delegated to the individual platforms. For example, actions such as refueling and visual identification (VID) are now simulated without user intervention. This allows the person managing Blue forces to focus on tactical decision-making, such as the command and control of assets. Also, smart-platform behavior results in a more formidable opponent in Red force, which now behaves more intelligently. If their detections and weapons status warrant, Red-force platforms will leave their previously specified paths to pursue Blue-force platforms. This creates much more challenging and demanding raids for the Blue force to defend against. The recent addition of Green-force platforms to this version of BATMAN & ROBIN further simulates uncertainty, confusion, and the "fog-of-war."

3.1 Knowledge Representation and Software Design Alternatives

Since what constitutes artificially intelligent or smart-platform behavior is debatable, it was crucial that the knowledge representation and software designs selected be *flexible* enough so that they could be enhanced and refined as necessary. Additionally, the designs for smart-platform behavior had to meet several other criteria, namely:

- *compatible* in language and programming style with the rest of BATMAN & ROBIN;
- *efficient*, so as not to degrade the performance of other BATMAN & ROBIN computer models (now and in the future), and user interaction and response time;
- *maintainable* by software engineers and users;
- *portable* to other hardware platforms that may host BATMAN & ROBIN (now and in

the future); and

- *explicit*, so that the assumptions and constraints inherent in artificially intelligent platform behavior are apparent.

Using these criteria, many designs were considered in our selection of a Knowledge-Based Finite-State Automata (FSA) model to implement smart-platform behavior in BATMAN & ROBIN. The following section discusses several design alternatives that were considered, and our reasons for rejecting those we did.

3.1.1 LISP and Expert-System Shells

We rejected building a LISP expert system because it was judged too slow, thus decreasing BATMAN's performance substantially; too time consuming, thus increasing development and maintenance costs; and too incompatible, thus making it difficult to integrate with BATMAN & ROBIN as well as computer models written in different languages (Butler, Hodil, & Richardson, 1988). We also considered using off-the-shelf expert system shells, but decided against them because they are expensive, difficult to integrate into BATMAN & ROBIN, and degrading to BATMAN's performance.

3.1.2 Procedural Decision Method

One plausible software design was to use the procedural-decision method. Under this scheme, each platform would have a function dictating how the platform should behave under any given set of circumstances. The advantage of this method is that it is easy for programmers to understand and maintain due to its use of conventional programming techniques. However, there are several disadvantages with the procedural-decision method: (a) all the decision making would be hard-coded in software, thus making the smart-platform logic difficult for nonprogrammers to comprehend; (b) changing the smart-platform logic under this circumstance would require a major programming effort and the funding to support it; and (c) the software could easily become unmanageable because this approach lacks a uniform structure.

3.1.3 Weighted-Influence Model

Another possibility was to use a weighted-influence model (Holtzman, 1989) to simulate intelligent decision making. Influence diagrams could be created for all platform types depicting all the factors in a simulated scenario that the platform would consider, e.g., defense warnings, weapons status, or sensor detections; and their relative importance and relationship to one another. Since this method does not impose traditional logical constraints on the decision-making process, the behavior of BATMAN platforms could be simulated with higher fidelity. By using weighted randomization in the model, "human-like" behavior could be created with the same situation resulting in different action outcomes. However, since our experience with such systems is minimal, implementation would be very costly and risky.

3.1.4 Knowledge-Based System vs. Finite State Automata

We considered designing a knowledge-based system which would evaluate production rules to determine a platform's behavior. However, it became apparent that platforms have behavioral states which strongly influence their simulated decision making and subsequent action. This led us to consider the usage of FSAs, where platform behavior is defined by state. Changing conditions

cause transitions between states. However, the representation of these complex transitions necessitates the building of a rule base. Therefore, the best solution was judged to be a combination of a knowledge-based system and FSAs (Raeth, 1990).

3.1.5 Knowledge-Based Finite State Automata

A knowledge-based FSA model works in the following manner. A knowledge database defines platform FSAs that include action states and the rules which execute state transitions. Also, it associates each mission with a FSA. For example, the Blue Fighter Aircraft FSA might have states that include "Remain At CAP Station" and "Kill Enemy". The following rule, "Blue Air Kill Necessary", when evaluated to true, would cause or execute a state transition to "Kill Enemy":

Blue Air Kill Necessary => Hostile Action or (**Warning Red** and **Weapons Free**)

If the logic needs to be changed so that Blue fighters kill only when there is hostile action, one could simply remove the "or" portion of the rule.

The advantages to this design are (a) implementation and maintenance are straightforward, (b) modification of smart-platform behavior without additional programming, and (c) comprehension of platforms' decisions by examining the database. Because of its simplistic nature, the performance of this design in BATMAN & ROBIN should be exceptional. The disadvantages of knowledge-based FSAs are (a) they cannot handle complex weighted decision making, and (b) creating new FSAs and maintaining the smart-platforms simulation engine requires a comprehensive understanding of how the model works. The first problem could be overcome by gradually incorporating the weighted influence model mentioned above. The second is offset by providing adequate documentation of the model.

3.2 Determinants of Behavior

In BATMAN, there are three determinants affecting platform behavior: (a) the mission of the platform, (b) the external stimuli that the platform detects during the scenario, and (c) the platform's interaction with other members of its force, whether Blue or Red. For example, suppose a fighter's mission is to fly at a combat-air-patrol (CAP) station. Radar may indicate an unknown air detection. If this platform is the closest available Blue fighter, it would perform a VID.

In this version, the following simulated platform missions are incorporated in BATMAN & ROBIN:

- **Blue Fighter Aircraft**: Blue-force fighters assigned to CAPs or chainsaws.
- **Blue Air Surveillance**: Blue-force aircraft assigned to surface surveillance.
- **Blue Attack Aircraft**: Blue-force aircraft assigned anti-surface warfare (ASuW).
- **Blue Air ASW**: Blue-force aircraft assigned anti-submarine warfare (ASW).
- **Blue Air Tankers**: Blue-force tankers assigned to refuel other Blue-force aircraft.
- **Blue Ships**: Blue-force ships assigned to anti-air warfare (AAW), ASuW, and/or ASW.
- **Blue Submarines**: Blue-force submarines assigned to ASuW and/or ASW.

- **Red Fighter Aircraft**: Red-force fighters acting as specified in ROBIN for AAW.
- **Red Air Attack**: Red-force bombers acting as specified in ROBIN for ASuW.
- **Red Air Surveillance**: Red-force bombers acting as specified in ROBIN for surface surveillance.
- **Red Ships**: Red-force ships acting as specified in ROBIN for AAW, ASuW, and/or ASW.
- **Red Submarines**: Red-force submarines acting as specified in ROBIN for ASuW and/or ASW.
- **Red Anti-Ship Missiles**: Simulated anti-ship missiles (ASMs) in BATMAN.
- **Green Air**: Neutral-force aircraft acting as specified in ROBIN.
- **Green Ships**: Neutral-force ships acting as specified in ROBIN.
- **Green Submarines**: Neutral-force submarines acting as specified in ROBIN.

Also, in this version, the following external stimuli or user inputs affect a Blue platform's behavior or action:

- **Detections**: Radar, sonar, ESM, and VID are simulated.
- **Enemy Fire**: Hits from enemy fire cause battle-damage and can destroy a platform.
- **Warnings and Weapons Status**: Defense warnings and/or weapons status can affect a platform's behavior.
- **User Input**: The user still commands and controls Blue force during BATMAN.

Lastly, in this version, platform interaction deals primarily with determining how to distribute enemy detections among available platforms. This is discussed in Section 3.4, **Platform Interactions.**

3.3 Knowledge-Based FSAs

The actions associated with a specific mission are determined by its FSA, the rules used by the FSA, and the type of hostile platform associated with the mission. An FSA consists of a number of behavioral or action states connected by transitions. A platform can be in any state corresponding to the FSA for its mission. The state a platform is in determines what action it will take. For example, if a platform is in a VID state, it will attempt to visually identify or detect a target. Each transition in the FSA is associated with a rule. In every simulated cycle each platform has the opportunity to change state. This will occur if its current state has a transition with a rule that is evaluated as true. It will continue to change state until it reaches one with no transitions that can be traversed. Once it has reached such a state (whether it has changed states or not), it will perform the action associated with that state.

For each mission a hostile platform type can be specified. If this is the case, rules will only be

evaluated for the selected hostile platform type, except when hostile action or intent is manifested and the subject platform has the weapons necessary to engage.

The state transitions are checked in the order specified. For example, if the rule corresponding to transition 1 evaluates to true, no further transitions out of the current state will be checked. However, if it evaluates to false, transition 2 will be checked next. **In the FSA diagrams that follow, states denoted by a double circle are initial states for a platform. States denoted with a dashed circle are control states which are used merely to simplify the FSA and have no associated action.** Currently, the only example of this is "Low Fuel".

3.3.1 Action States

An action state characterizes a platform's ongoing behavior at any given time, such as flying a chainsaw or intercepting an unknown contact for VID. Some action states have a **target or object of interest**. For example, the object of interest in a VID state would be an unknown contact and the object of interest in a refueling state would be the tanker. A platform whose action state includes an object of interest is said to be **occupied**, otherwise the platform is unoccupied. Associated with each action state is a description of the behavior, the platform's speed while in the state, and the platform's range of interest while in the state.

Range of interest is a distance specified in the database for each platform type. When contacts lie within a platform's range of interest, the platform will consider taking action on them. A platform will ignore any contacts outside of its range of interest. Each platform has two ranges, a **surveillance range** and a **keep-out range**. Each action state will use one of these two ranges as the platform's range of interest. An action state where a platform is occupied will generally use the smaller keep-out range to avoid being distracted, unless the threat is imminent. Actions states where a platform is unoccupied will use the much larger surveillance range to maximize the number of contacts they can consider.

The following sections describe every action state implemented to date. A **state's action** is indicated by its name. Any **assumptions and constraints** concerning the execution of the action are listed here. The **speed** listed is used by a platform in that state, unless the platform is following a specified path or track or unless a hostile platform has a lock on it, in which case full power is used to evade or engage. The **range of interest** specifies whether the state uses the platform's surveillance range or keep-out range. A Blue platform's **state can be changed** by moving it towards a new location or towards another platform. This is covered in detail in the Section 3.3.2, **User Input**, but is listed briefly here for reference. The current action state of a Blue platform and its target are shown by the pop-up single-status display in BATMAN.

3.3.1.1 At/Return Station

Blue air platforms in this state remain at or return to their stations while looking for detections on which to take action. A station is an area or location to which the platform has been assigned by the user, e.g. CAPs, chainsaws, tanker stations, or surveillance areas.

Attributes:

Speed: max conserve

Range of interest: surveillance range

Assumptions: Moving a platform with this initial state effectively creates a new station for it.

Changing to this state: Move platform to new location. If the platform is pursuing another platform, it may need to be moved so that the current target would be out of sector range at the new location.

3.3.1.2 Continue ASW Mission

Blue air ASW platforms in this state remain at their current location or lay sonobuoys as directed by the user while searching for ASW detections on which to take action.

Attributes:

Speed: max conserve

Range of interest: surveillance range

Assumptions: None.

Changing to this state: Move platform to new location.

3.3.1.3 Continue Mission

Blue platforms in this state remain at their current location or move as directed by the user while looking for detections on which to take action.

Attributes:

Speed: max conserve

Range of interest: surveillance range

Assumptions: None.

Changing to this state: Move platform to new location.

3.3.1.4 Continue Track

Red platforms in this state follow their specified paths or tracks while looking for detections on which to take action. Green platforms in this state follow their tracks.

Attributes:

Speed: velocity specified for track in ROBIN

Range of interest: surveillance range

Assumptions: When a platform returns to its path, it returns to the end of the track vector it was following when it left. The speed used is that of this vector.

Changing to this state: N/A

3.3.1.5 Visual Identification

Blue air platforms in this state attempt to intercept a previously specified unknown contact to get a VID.

Attributes:

Speed: full power

Range of interest: keep-out range

Assumptions: VID does not take platform heading or altitude into consideration. A positive ID can always be made when within the visual range specified in the database.

Changing to this state: Move platform to a detection which has not been positively IDed.

3.3.1.6 Escort Threat

Blue fighters in this state attempt to intercept and follow a user-designated, previously identified aircraft.

Attributes:

Speed: full power until target reached, then target's speed is matched

Range of interest: keep-out range

Assumptions: If the platform is matching the target's speed and has a velocity in between max conserve and full power, the fuel consumption is linearly interpolated based on its maximum and minimum fuel consumptions.

Changing to this state: Move platform to a detection which has been positively IDed.

3.3.1.7 Intercept

Blue platforms in this state attempt to intercept a user-designated contact and stop when in VID range.

Attributes:

Speed: full power

Range of interest: keep-out range

Assumptions: None.

Changing to this state: Move platform to a detection.

3.3.1.8 Kill (Platform)

Red or Blue platforms in this state will continue on course and fire on a previously selected target.

Attributes:

Speed: max conserve

Range of interest: keep-out range

Assumptions: The following assumptions also apply to Close Kill described below. If a platform needs to fire weapons, it will turn on both search and target acquisition radar when they are not already on. Weapons are always fired longest range first, then the next longest, and so on. Weapons never accidentally hit an unintentional target. Platforms can only shoot at one target at a time and fire once per simulation cycle. Damage to a platform does not diminish any of its capabilities. Weapons firing disregards platform heading. Currently, launch acceptability regions for deciding when to fire weapons are modeled using only ra .ge and altitude.

Changing to this state: Do intercept and platform will transition to Kill if possible.

3.3.1.9 Close Kill

Red or Blue platforms in this state attempt to intercept and fire on a previously designated target.

Attributes:

Speed: full power

Range of interest: keep-out range

Assumptions: When aircraft close in on ships, they stop moving in when doing so would put them within range of the longest range weapon the surface combatant could use against them. Red force platforms will leave their tracks or paths when in this state. Also, see Kill assumptions above in addition to these.

Changing to this state: Do intercept and platform will transition to Close Kill if possible.

3.3.1.10 Refuel

Blue air platforms in this state attempt to rendezvous with a tanker and refuel.

Attributes:

Speed: max conserve

Range of interest: keep-out range

Assumptions: Only Blue aircraft consume fuel.

Changing to this state: Move platform to tanker.

3.3.1.11 Give Fuel

A Blue air tanker in this state will attempt to rendezvous with another Blue aircraft that has communicated low fuel, and refuel it.

Attributes:

Speed: max conserve

Range of interest: keep-out range

Assumptions: Can only give fuel to one platform at a time

Changing to this state: Move platform that needs fuel to tanker.

3.3.1.12 Return to Base

Blue aircraft in this state return to their home bases, i.e., carriers, ships, or air fields. Red platforms in this state return to their initiation points, air fields or ports, and then disappear from the display.

Attributes:

Speed: max conserve

Range of interest: keep-out range

Assumptions: If Blue air platforms home bases are destroyed, they cannot land anywhere else and will splash. Red platforms use the speed of their last track when returning unless a Blue platform has a lock on them.

Changing to this state: Move platform to mother ship or base.

3.3.1.13 Missile State

Missiles move to their target at specifiable speed.

Attributes:

Speed: full power

Range of interest: None.

Assumptions: None.

Changing to this state: The missile is launched by another platform.

3.3.2 User Input

The move function is now used to supplement or override smart-platforms behavior during the simulation. Moving a platform can cause three things to happen: (a) if a location is selected, the platform moves towards it; (b) if another platform is selected and the platform can enter a state which applies to it, the platform's state will change and the other platform will become the object of the new state's action; and (c) if the move requested causes any rules to be violated, these rules will be ignored as long as appropriate.

Moving a platform to a new location can serve one of the following purposes:

- If the platform's current state is **At / Return Station**, a new station will be created and the platform will move to it.
- The platform will reevaluate its situation, often abandoning its current action, particularly if the platform has a station and is placed out of range of the current target.

- If a platform is returning to base because it is low on fuel, out of weapons or out of sonobuoys, the corresponding rule will be ignored until the platform is resupplied.

Moving an aircraft to its home base causes it to land. Moving an aircraft to a tanker refuels it. This will cause the tanker to abandon any other platforms it is refueling, and move towards the selected aircraft.

Moving a platform to a detected bogie, causes the platform to attempt to intercept the bogie, abandoning any previous pursuit. The result of this intercept depends on the current Blue warning level and weapons status, and the level of the detection, e.g., unknown or hostile. First, consider the case where weapons status is "tight." If the detection has not been positively identified and the platform is capable of doing a VID, it will do it. If it has been positively identified and the platform is capable of escorting it, it will. If the platform can do neither of these two things, it will intercept the platform, stopping when within VID range. In the case where the warning level is "red" and weapons status is "free" against the detection, the platform will behave as before, except that, the platform will attempt to kill the detected platform, if possible.

The **Reeval Target** rules will be ignored for targets designated by moving a Blue platform to a detection, unless another platform has a lock on the Blue platform itself. If an attack aircraft with no stand-off weapons is moved to a ship, the **Close In Dangerous** rule will be ignored until the aircraft lands. If the detection is outside the platform's sector or current area of interest, range checks will be ignored as long as the detection is being pursued.

3.3.3 General Rules

The decision to transition to a new state is determined by evaluating the rule associated with the transition. Rules fall into two categories: basic rules and composite rules. **Basic rules** are defined by stating a condition in the simulation which causes them to be true. **Composite rules** are defined in terms of other rules, as modified by the logical operators "and", "or" and "not".

Example:

Basic Rule: "**Visual Ided** : True if target platform positively identified."

Composite Rule: "**No Vid** => not **Visual Ided**"

Some rules can be overridden by the user. In this case, the rule is made up of a basic rule and an override rule.

Example:

Low Fuel Basic : True if this platform has just enough fuel to return to base. In the case of tankers, this is true when there is no remaining give fuel.

Use Low Fuel : True if the low fuel check has not been disabled.

Low Fuel => **Low Fuel Basic** and **Use Low Fuel**

Thus, references to the **Low Fuel** rule will only be used if not disabled by the user.

Some rules vary depending on the mission of the platform. These are called variant rules.

Example:

Kill Special Restrictions => Variant

Blue Fighter Aircraft, Blue Air Surveillance : **Blue Air Kill**

Others: True, no restrictions

In this example, the **Blue Air Kill** rule is used for platforms with the Blue Fighter Aircraft and the Blue Air Surveillance missions. For all other platforms, the rule always evaluates to true.

This section lists all the general rules which are shared by several missions. Additional rules are listed with the missions themselves. Rules which appear in the FSAs are capitalized. Rules are ordered by functionality. Rules of the same function are ordered by complexity, with the simpler rules appearing first. All rule names are printed in boldface. In the following sections, "target platform" refers to the detected platform for which the rule is being evaluated.

3.3.3.1 Identification Rules

Visual Ided : True if target platform positively identified.

Enemy Platform : True if target platform has been identified as the enemy through ESM or other positive means.

Unknown Platform : True if target platform has been identified, but is not known to be hostile.

Enemy Is Ship : True if target platform is a ship; this is always known.

Enemy Is Sub : True if target platform is a submarine; this is always known.

Enemy Or Unknown => **Enemy Platform** or **Unknown Platform**

Not Enemy Or Unknown => not **Enemy Or Unknown**

Enemy Is Not Air => **Enemy Is Ship** or **Enemy Is Sub**

3.3.3.2 Intercept Rules

Intercept Possible: True if platform can intercept target platform, assuming target maintains present course and speed. Air-to-air intercepts are only considered possible if they would take fifteen minutes or less.

Intercept Not Possible => not **Intercept Possible**

Target Reachable => **Intercept Possible** or **In Weapons Range**

Target Not Reachable => not **Target Reachable**

Target Reached : True if within VID range of target.

INTERCEPT OVER => **Target Reached** or **Enemy Gone** or **Intercept Not Possible**

Intercept Restrictions => Variant

Blue Sub, Red Ship, Red Sub : **Enemy Is Not Air**

Blue Ship : False, never intercept

Others : True, always intercept

3.3.3.3 Combat Rules

Hostile Action : True if target platform has exhibited hostile action. In this case, sometime during the simulated battle it has fired weapons or turned on target acquisition radar. This is always known.

No Hostile Action => not **Hostile Action**

Not enough attackers : True if not enough platforms are going after the target platform as specified by the database, or, if this platform is the closest to the target and some of the other attackers do not yet have their target acquisition radar on.

Appropriate Weapons : True if this platform has weapons that can be used against the target platform.

No Appropriate Weapons => not **Appropriate Weapons**

In Weapons Range : True if this platform is within weapons range of the target platform.

Locked On Me : True if the target platform is the closest with a lock on this platform.

My TAR Not On : True if this platform does not have its target acquisition radar on.

Warning Red : True if this platform's force is at warning red against the target platform's type.

Weapons Free : True if this platform's force has weapons free against the target platform's type.

Warning Red Free => **Weapons Free** and **Warning Red**

Kill Special Restrictions => Variant

Blue Fighter Aircraft, Blue Air Surveillance : **Blue Air Kill**

Others: True, no restrictions

Kill Restrictions => **Warning Red Free** and **Enemy Or Unknown** and **Kill Special Restrictions**

Kill Necessary => Hostile Action or **Kill Restrictions**

Should Kill => Appropriate Weapons and **Kill Necessary**

Close In Dangerous Basic : True if the target platform is of a different type than the detecting platform, e.g., air vs. surface, and the target platform is known to carry longer range weapons against the detecting platform than it has against the target.

Close In Dangerous => Close In Dangerous Basic and **Use Dangerous**

Close In Not Dangerous => not **Close In Dangerous**

Reeval Target Plat => Use Reeval Target Plat or **Locked On Me**

Higher Priority In Place Kill => Reeval Target Plat and **In Weapons Range** and **Should Kill**

Higher Priority Close In Kill => Reeval Target Plat and **Intercept Restrictions** and **Close In Not Dangerous** and **Should Kill**

CHANGE TARGET IN PLACE => My TAR Not On and **Higher Priority In Place Kill**

CHANGE TARGET CLOSE IN => My TAR Not On and **Higher Priority Close In Kill**

DO IN PLACE KILL => In Weapons Range and **Should Kill**

DO CLOSE IN KILL => Intercept Restrictions and **Close In Not Dangerous** and **Should Kill and Not Enough Attackers** and **Target Reachable**

SHOULD KILL OR CHANGE TARGET CLOSE IN => Should Kill or **Change Target Close In**

3.3.3.4 Abort Mission Rules

Low Fuel Basic : True if this platform has just enough fuel to return to base. In the case of tankers this is true when no give fuel remains.

LOW FUEL => Low Fuel Basic and **Use Low Fuel**

No Weapons Basic : True if this platform has no more weapons needed for its mission.

NO WEAPONS => No Weapons Basic and **Use No Weapons**

NO WEAPONS OR FUEL => No Weapons or **Low Fuel**

NO WEAPONS OR FUEL OR TOO DANGEROUS => No Weapons Or Fuel or **Close In Dangerous**

3.3.3.5 Abort Kill Rules

ENEMY GONE : True if detection of the target platform has been lost or is outside its

range, and range has not been overridden by the user.

Target Not Hostile => **Not Enemy Or Unknown** and **No Hostile Action**

Kill Over => **Enemy Gone** or **No Appropriate Weapons** or **Target Not Hostile**

Not In Weapons Range => not **In Weapons Range**

IN PLACE KILL OVER => **Not In Weapons Range** or **Kill Over**

Other Plats Closer Basic : In the case of a close-in-kill, this is true if this platform does not yet have target acquisition radar on and a specified number of other platforms do. In all other cases, this is true if another platform is intercepting the target platform and is closer.

Other Plats Closer => **Use Reeval Target Plat** and **Other Plats Closer Basic**

CLOSE IN KILL OVER => **Other Plats Closer** or **Target Not Reachable** or **Kill Over**

3.3.3.6 Refueling Rules

TANKER AVAILABLE : If platform is already rendezvousing with a tanker, this is true if tanker is still available; otherwise, this is true if there is a tanker that the platform can reach, and the tanker has enough give fuel to provide half a tank.

No Tanker Available => not **Tanker Available**

Refueled : True if the platform has a full tank of fuel.

REFUEL OVER => **Refueled** or **No Tanker Available**

3.3.3.7 User Intervention Rules

Use Low Fuel : True if the low fuel check has not been disabled.

Use No Weapons : True if the no weapons check has not been disabled.

Use Dangerous : True if the dangerous check has not been disabled.

Use Reeval Target Plat : True if the reevaluation of target platforms has not been disabled.

3.3.4 Platform Missions

This section contains descriptions of the simulated missions implemented in BATMAN & ROBIN. Included with each mission description is a list of platforms that perform the mission, the FSA, any specific rules used by the FSA, and any assumptions made pertaining to the mission. To facilitate reading the documentation, it was organized so that every mission begins at the top of a new page.

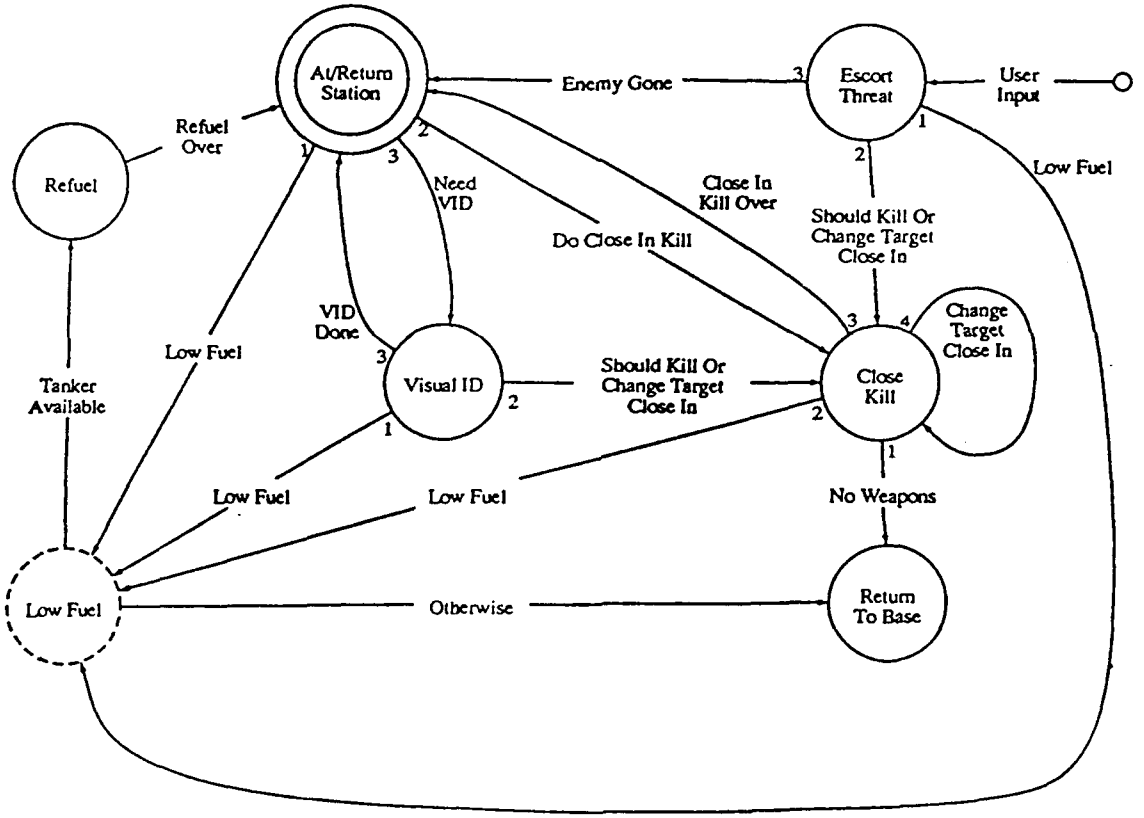
3.3.4.1 Blue Fighter Aircraft

Mission Description:

Patrol sector against Red aircraft, either at CAP station or chainsaw.

Platform Types: F-14, F-15, F-16, F-18, F-4E, and AH-1T.

FSA:



Assumptions:

None.

Rules Specific to FSA:

Hostile ESM : True if hostile radar emissions have been received from target platform.

Vid Or ESM => Visual Ided or **Hostile ESM**

Out Of My Range Basic : True if target platform is not within sector radius of CAP station. If platform is on a chainsaw, station is considered to be closest end of chainsaw.

Use Range : True if the range check has not been disabled.

Out Of My Range => Out Of My Range Basic and **Use Range**

In My Range => not **Out Of My Range**

Blue Air Kill => Enemy Platform and **In My Range** and **Vid Or ESM**

Not Being Visual IDed : True if no closer platform is currently attempting to VID the target platform.

No VID => not **Visual IDed**

Vid Warning => Variant

Blue Fighter Aircraft : **Warning Red Free**

Blue Air Surveillance : **Warning Red**

Others : False

ESM VID Warning => Hostile ESM and **VID Warning**

Not ESM VID Warning => not **ESM VID Warning**

NEED VID => No VID and **Not Being VIDed** and **No Hostile Action** and **In My Range** and **Not ESM VID Warning** and **Intercept Possible**

VID DONE => Visual IDed or **Enemy Gone** or **Out Of My Range** or **Hostile Action** or **Other Plats Closer** or **Intercept Not Possible**

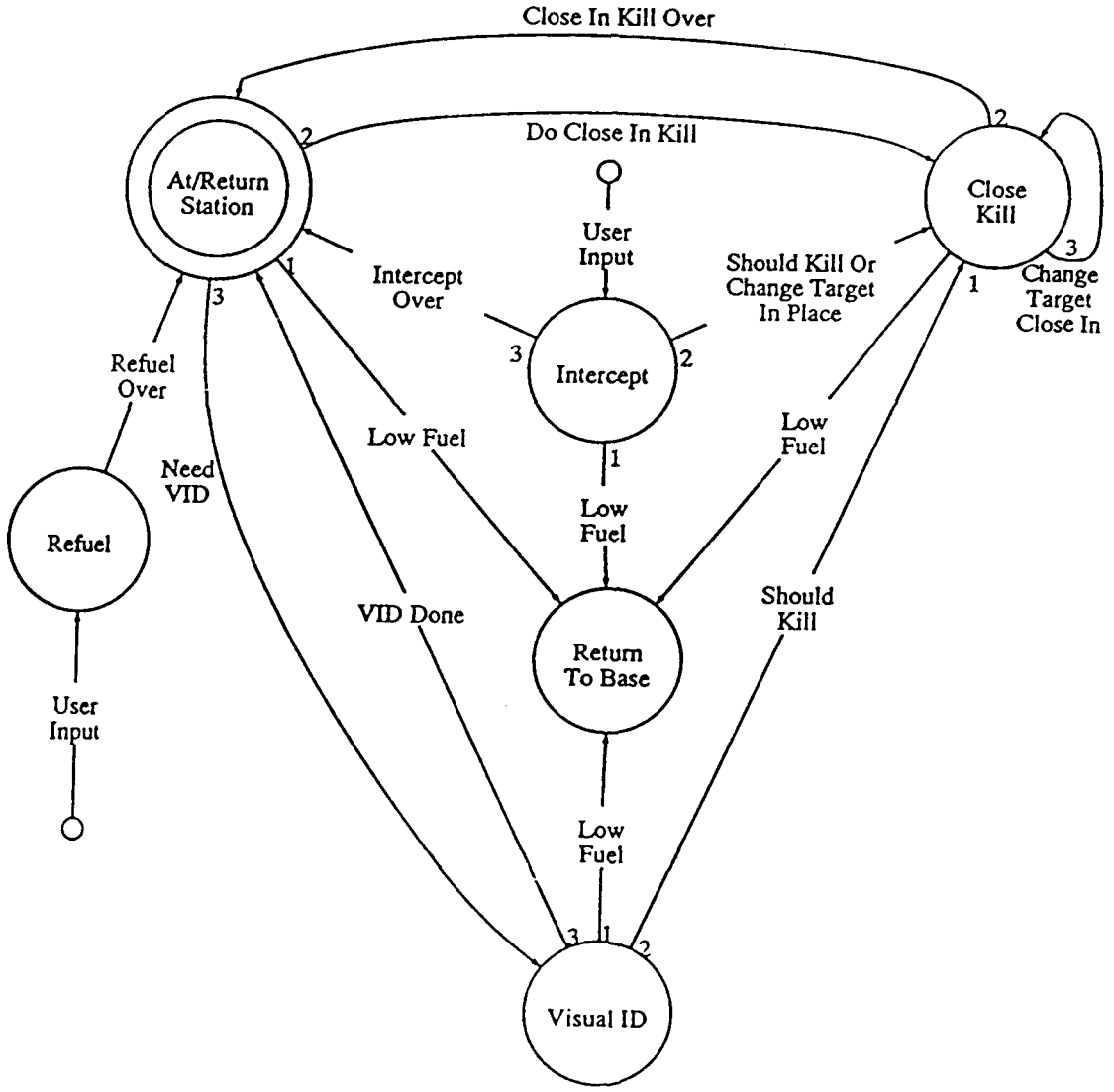
3.3.4.2 Blue Air Surveillance

Mission Description:

Patrol surveillance area against Red ships.

Platform Types: E-2C, EA-6B, OV-10A, E-3B, UH-1N, CH-46, and CH-53.

FSA:



Assumptions:

None.

Rules Specific to FSA:

Hostile ESM : True if hostile radar emissions have been received from target platform.

VID Or ESM => Visual IDed or **Hostile ESM**

Out Of My Range Basic : True if target platform is not within sector radius of CAP station. If platform is on a chainsaw, station is considered to be closest end of chainsaw.

Use Range : True if the range check has not been disabled.

Out Of My Range => Out Of My Range Basic and **Use Range**

In My Range => not **Out Of My Range**

Blue Air Kill => Enemy Platform and **In My Range** and **VID Or ESM**

Not Being Visual IDed : True if no closer platform is currently attempting to VID the target platform.

No VID => not **Visual IDed**

VID Warning => Variant

Blue Fighter Aircraft : **Warning Red Free**

Blue Air Surveillance : **Warning Red**

Others : False

ESM VID Warning => Hostile ESM and **VID Warning**

Not ESM VID Warning => not **ESM VID Warning**

NEED VID => No VID and **Not Being VIDed** and **No Hostile Action** and **In My Range** and **Not ESM VID Warning** and **Intercept Possible**

VID DONE => Visual IDed or **Enemy Gone** or **Out Of My Range** or **Hostile Action** or **Other Plats Closer** or **Intercept Not Possible**

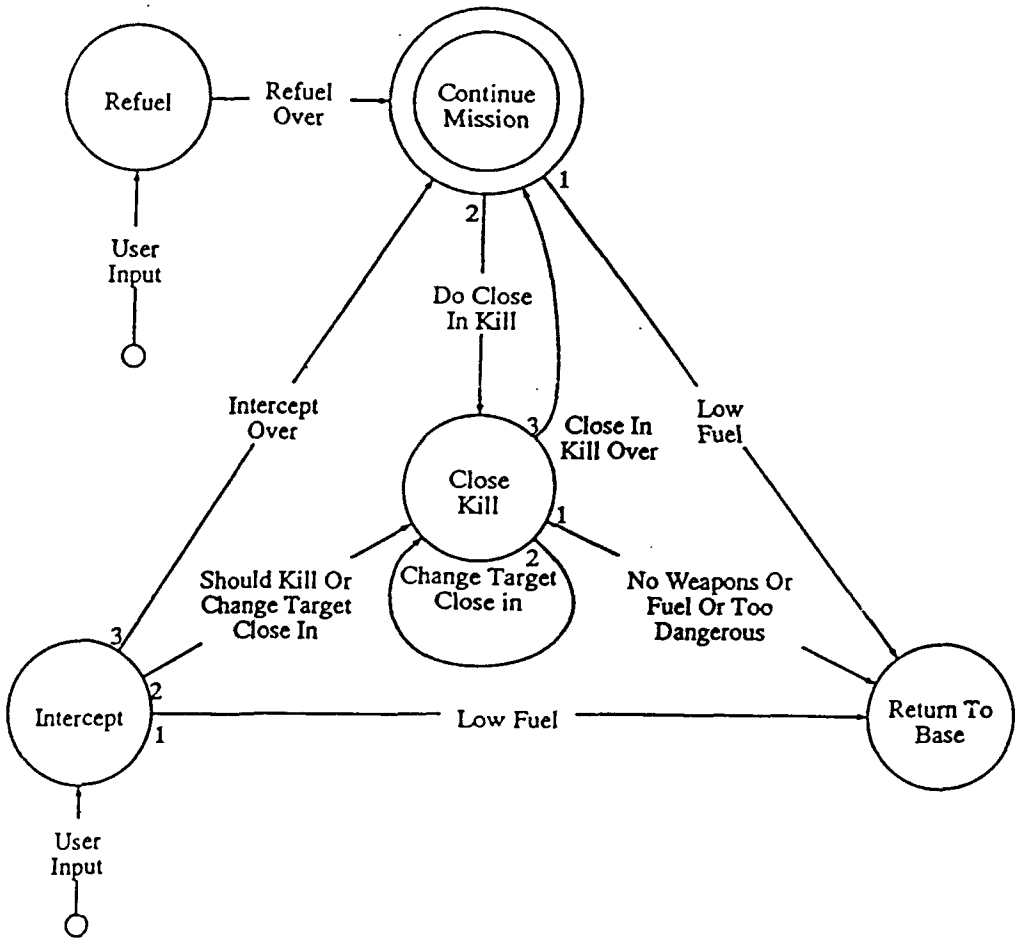
3.3.4.3 Blue Attack Aircraft

Mission Description:

Attack Red ships.

Platform Types: A-6, A-7, A-18, AV-8B, and A-4M.

FSA:



Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

Assumptions:

None.

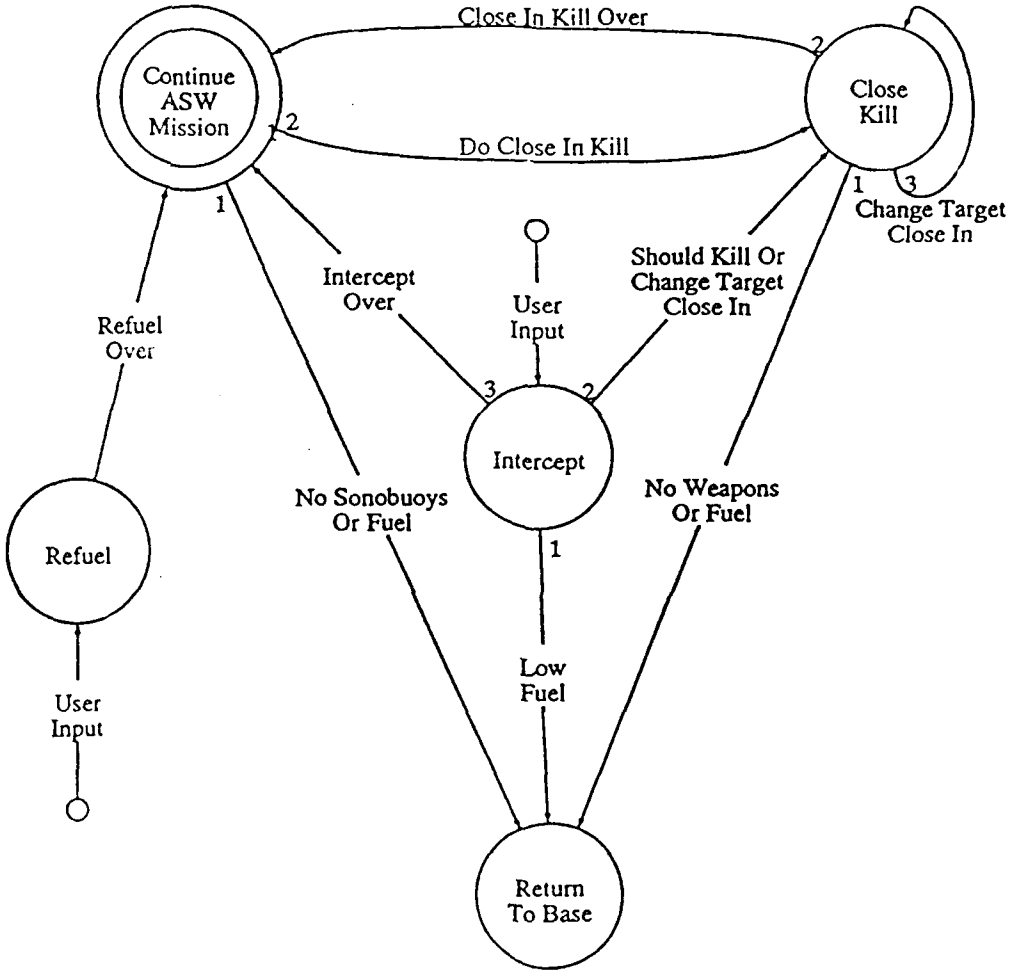
3.3.4.4 Blue Air ASW

Mission Description:

Locate and attack Red submarines.

Platform Types: P-3C, S-3B, SH-2F, SH-3H, and SH-60B.

FSA:



Assumptions:

A passive sonar cross detection is considered a positive identification.

Rules Specific to FSA:

No Sonobuoys Basic: True if this platform is out of sonobuoys.

Use No Sonobuoys: True if the no sonobuoys check has not been disabled.

No Sonobuoys => No Sonobuoys Basic and **Use No Sonobuoys**

NO SONOBUOYS OR FUEL => Low Fuel or **No Sonobuoys**

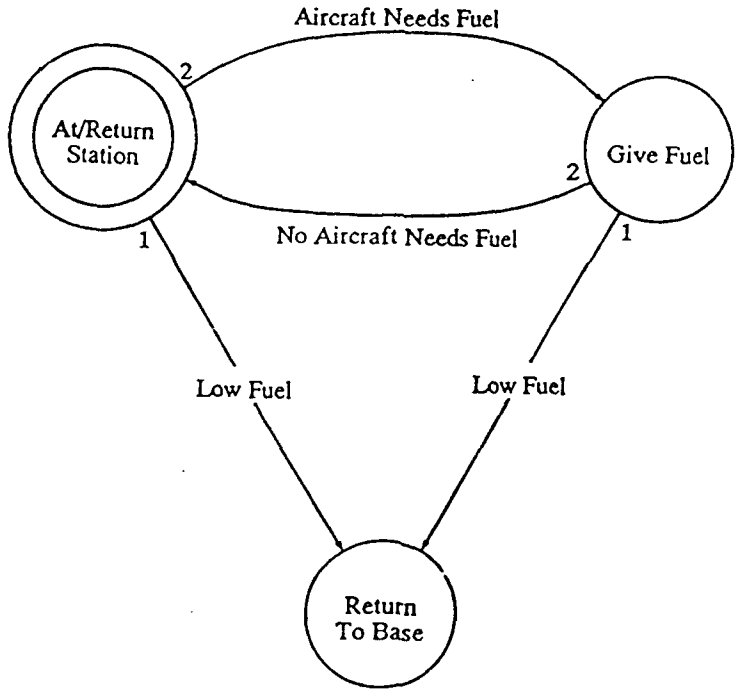
3.3.4.5 Blue Air Tankers

Mission Description:

Refuel other Blue aircraft, including those of other task forces, meeting them half way.

Platform Types: KA-6D, KC-10, KC-130, and KC-135.

FSA:



Assumptions:

A tanker refuels itself completely from its give fuel, unless it is enroute to refuel another platform, in which case it takes only what it needs to return to base.

Rules Specific to FSA:

AIRCRAFT NEEDS FUEL : True if an aircraft has requested refueling.

NO AIRCRAFT NEEDS FUEL => not **Aircraft Needs Fuel**

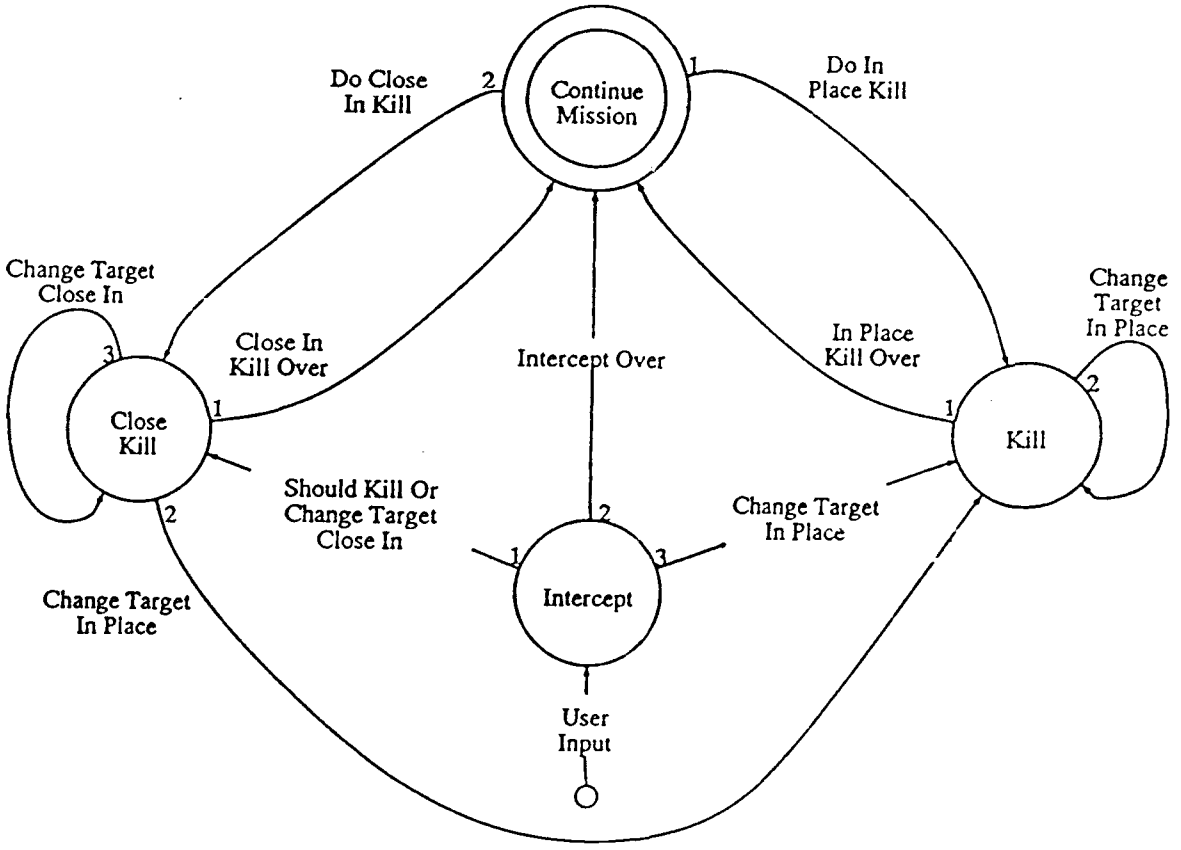
3.3.4.6 Blue Ships

Mission Description:

Attack any Red air, surface, and subsurface platforms in weapons range.

Platform Types: Nimitz, Ticonderoga, California, Iowa, Knox, Kidd, Perry, Spruance, Leahy, Belknap, and Tarawa.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

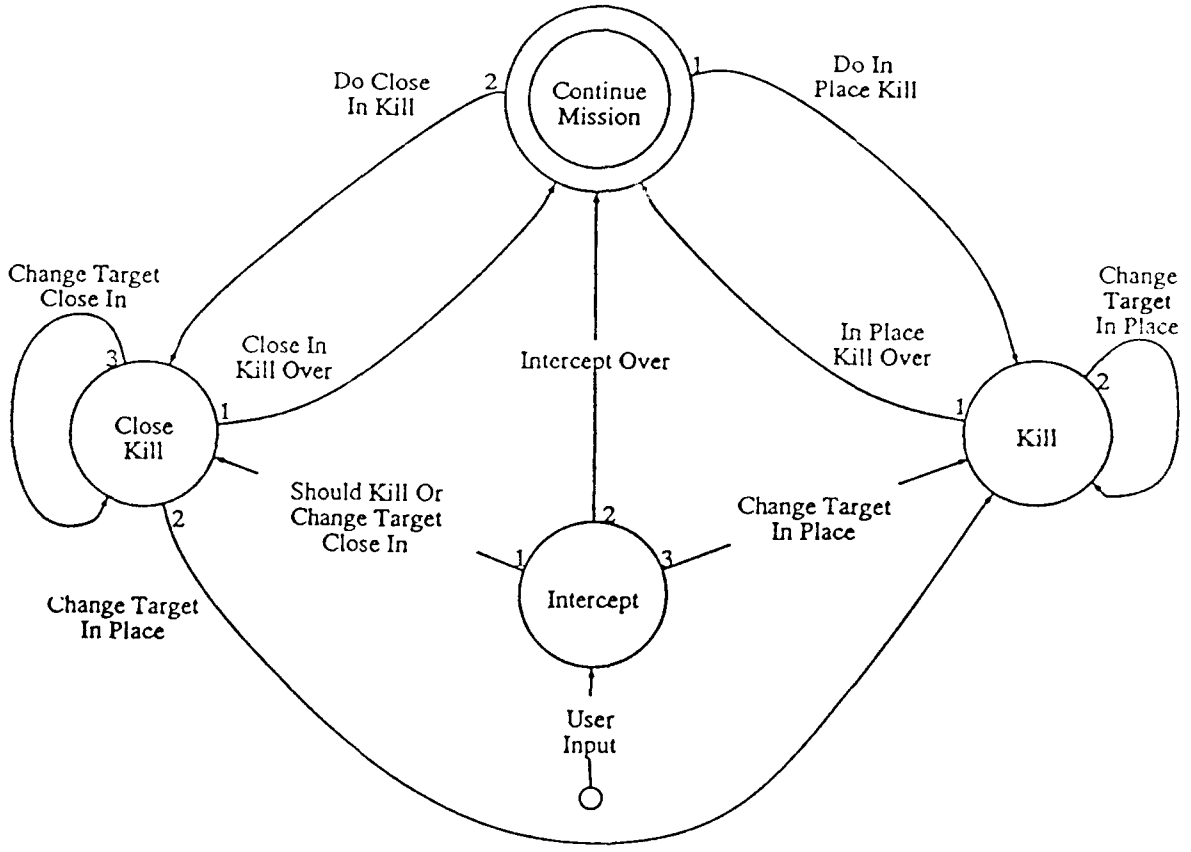
3.3.4.7 Blue Submarines

Mission Description:

Attack Red submarines or ships detected.

Platform Types: Los Angeles.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules**.

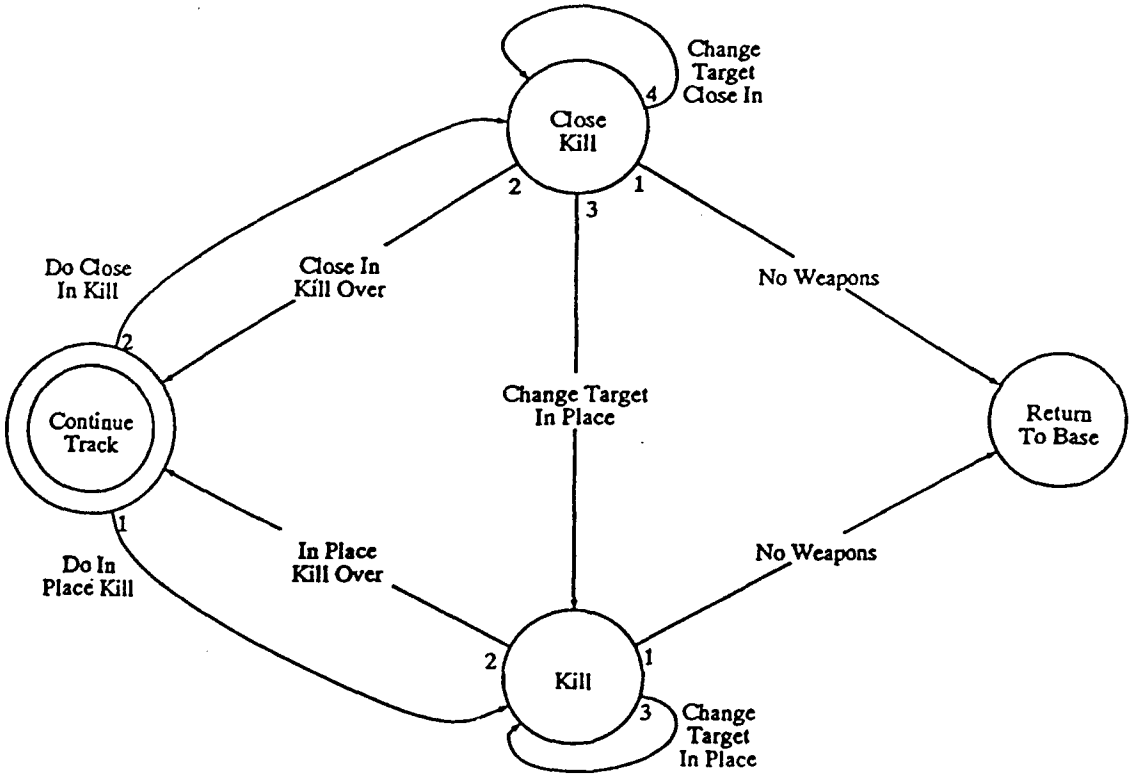
3.3.4.8 Red Fighter Aircraft

Mission Description:

Attack Blue aircraft.

Platform Types: MiG-19, MiG-21, MiG-23, MiG-25, MiG-27, MiG-29, MiG-31, Su-7, Su-9, Su-15, and Su-24.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

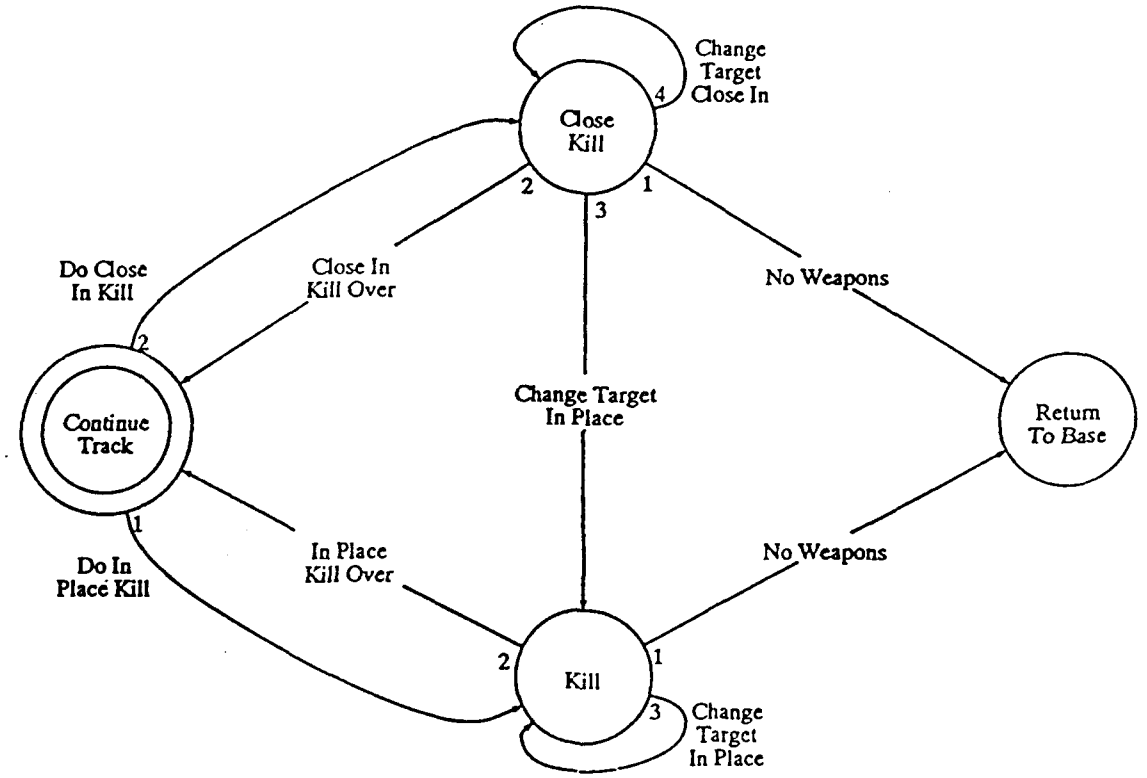
3.3.4.9 Red Attack Aircraft

Mission Description:

Attack Blue ships.

Platform Types: Bear, Badger, Blinder, Backfire, Blackjack, MiG-23, MiG-27, Su-7, and Su-24.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

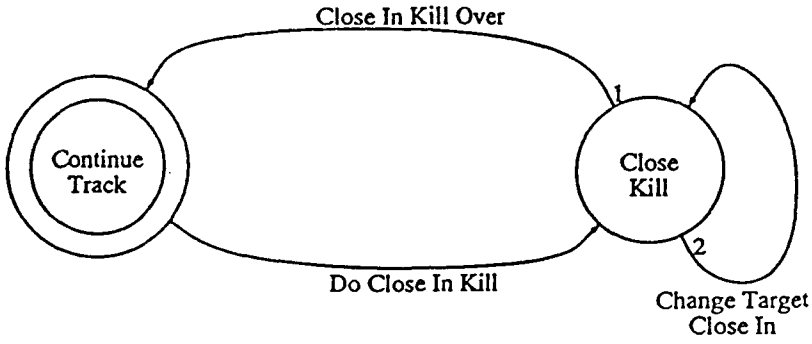
3.3.4.10 Red Air Surveillance

Mission Description:

Find Blue ships.

Platform Types: Bear and Mainstay.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

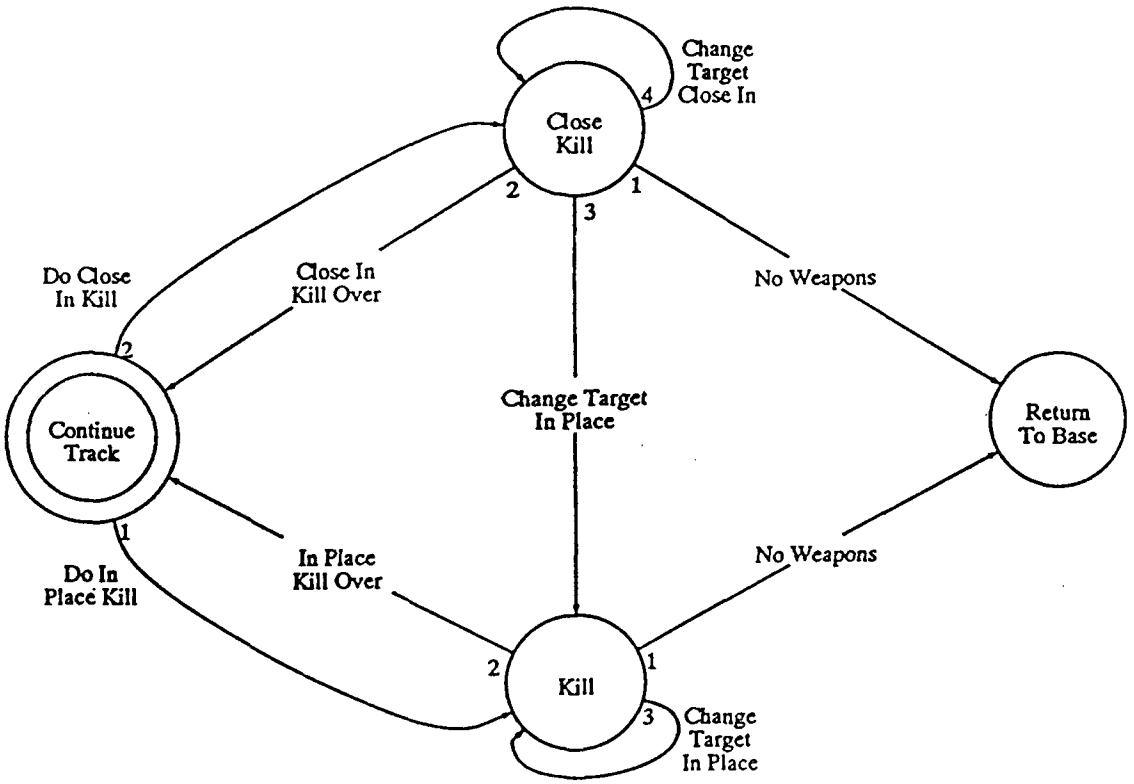
3.3.4.11 Red Ships

Mission Description:

Attack Blue air, surface, and subsurface platforms detected.

Platform Types: Kara, Kashin, Kiev, Kirov, Kresta, Krivak, Kynda, Moskva, Slava, Sovremeny, and Udaloy.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

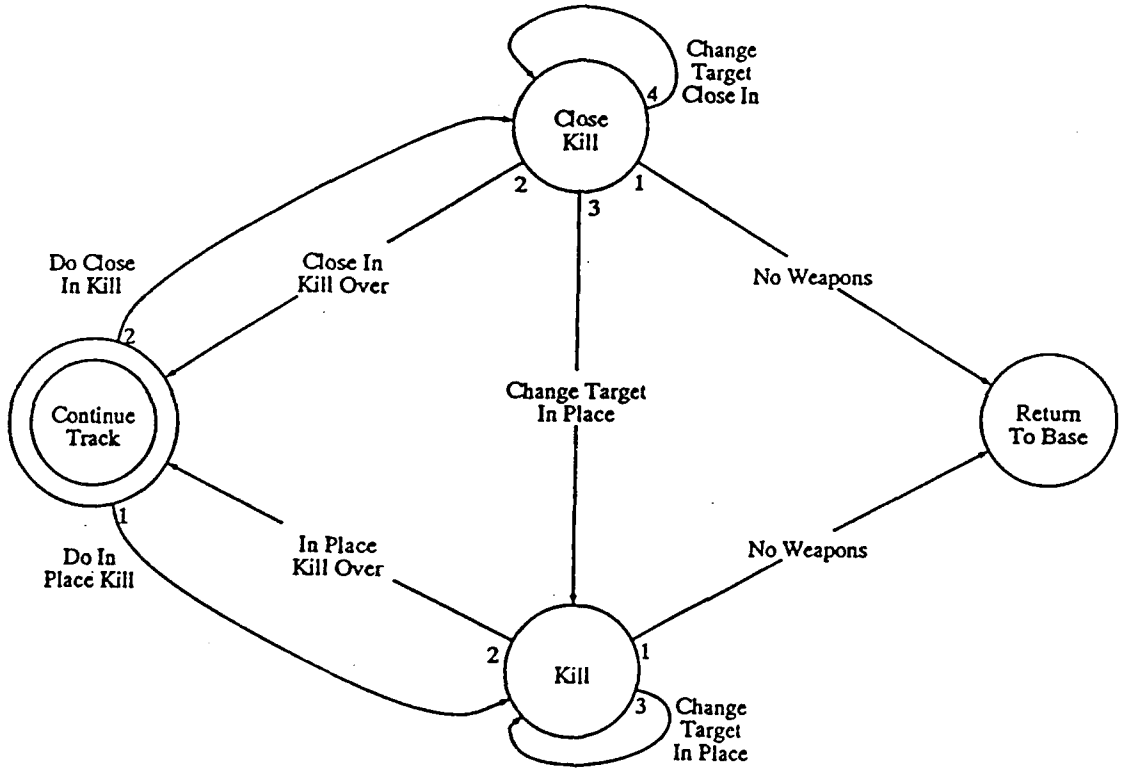
3.3.4.12 Red Submarines

Mission Description:

Attack Blue surface and subsurface platforms detected.

Platform Types: Charlie II, Delta-I, Echo II, Foxtrot, and Victor III.

FSA:



Assumptions:

None.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

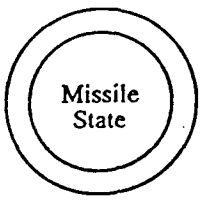
3.3.4.13 Red Anti-Ship Missiles (ASMs)

Mission Description:

Destroy Blue ships. No decision-making is employed by the ASMs themselves.

ASMs: AS-2, AS-4, AS-5, AS-6, and AS-15.

FSA:



Assumptions:

Detection is not modeled for ASMs and they are automatically able to locate their targets.

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

3.3.4.14 Green Air, Green Ships, and Green Submarines

Mission Description:

Green or neutral platforms follow their tracks or paths and activate their sensors as specified in ROBIN. Green platforms identify themselves and make no decisions in BATMAN.

Platform Types: All Green platforms.

FSA:

Assumptions:

Green platforms do not fire weapons. Neutral platforms identify themselves when the following conditions are true (Warning level refers to that of the Blue force):

Green Aircraft:

Warning Red and within 100nm of air search radar

Warning Yellow and within 50nm of air search radar

Warning White and within 20nm of air search radar

Green Ships:

Warning Red and within 20nm of combatant

Green Submarines:

Warning Red or Yellow and actively searched

Rules Specific to FSA:

None. See Section 3.3.3, **General Rules.**

3.4 Platform Interactions

Important aspects of intelligent platform behavior are determining who pursues whom and who will fire on whom. When making these determinations, we have adopted the following guidelines for Blue and Red forces:

- Platforms should attack or pursue the contacts that are closest to them.
- Platforms of the same force should not "gang-up" on, or all pursue a single threat while ignoring others.
- Platforms should break-off an attack or pursuit to defend themselves from a more imminent threat, e.g., a Red platform has a lock on a Blue platform.
- Platforms should relinquish a pursuit to a closer platform of the same force.

To implement these guidelines, we assumed that platforms of a particular force, Blue, Red, or Green, can all communicate with one another, and that this communication happens instantaneously. All detections are communicated to all other members of a force. Also, information about who is shooting whom, who is pursuing whom, and the locations of all members of a force is available to all of its members. Rules in the FSA for each platform use the following information about any detection within a platform's area of interest to determine which contact, if any, the platform will pursue:

- Detection level (unknown, neutral or hostile ESM, ESM or positive ID).
- Location.
- Hostile Action: the contact has or had a weapons lock on a member of the force, or has fired weapons.
- Current number of attackers.
- Number of friendly platforms with a weapons lock on the detected platform or contact.
- Closest interceptor.
- Closest interceptor with a weapons lock on the contact.
- Whether or not the contact has a weapons lock on the detecting platform. (A platform always knows when a contact has a weapons lock on it, but not all weapons require target acquisition radar.)

In general, a platform chooses a target to attack from the list of contacts in its area of interest and sector based on the priorities which follow. For the exact criteria for target selection, see the FSA and rules for the appropriate mission type under Section 3.3.4, **Platform Missions**.

- The closest platform with a lock on it will be chosen first.
- The closest platform having exhibited hostile action, which is not already being attacked by its specifiable maximum number of attackers, will be chosen next.

- The closest, highest priority platform which is not already being attacked by its maximum number of attackers will be chosen next.

"Maximum number of attackers" in the above list refers to a value specified in the database for each platform type. Restricting the number of attackers for each target prevents the ganging-up problem mentioned earlier, and forces a more reasonable pairing-up of targets and attackers. An attacker can override this restriction if he is not preoccupied, and is already within weapons range of the target.

In addition to attacking, Blue platforms must often intercept and VID unknown contacts. Blue platforms will chose the closest contact that is not being intercepted by a member of the same force which is closer to the contact. For the exact criteria, see the FSA and rules for the appropriate mission type in Section 3.3.4, **Platform Missions.**

Part III: Software Design

4.0 Purpose and Scope

This part of the documentation provides an overview of BATMAN & ROBIN software. It includes descriptions and diagrams of the system's software packages, data structures, and interfaces, and can be used as an aid by those interested in modifying and enhancing the system. The next section provides an introduction to the adopted software-design philosophy, with some guidelines to follow when modifying the software. The remaining sections in this part describe for BATMAN & ROBIN each software component, global data structures, software packages, and software interfaces to other computer models. For implementation details on specific BATMAN & ROBIN packages, refer to the commented code in the directory: **/nprdc/wargame/batman.**

5.0 Introduction

To help provide a better understanding of BATMAN & ROBIN software, this section outlines the design philosophy and system guidelines.

5.1 Design Philosophy

The intent of the human-computer interface is to provide intuitive ease and flexibility in constructing and gaming tactical scenarios. To achieve these goals and allow for future system expansion, much effort has gone into creating generic data structures, modularized software, and functional standards in a object-oriented-programming style that provide a natural mapping between the system's human interface and the code-level design. These principles of the interface and the software packages should be maintained by all persons interested in contributing to this system.

5.2 Software Caveats

Before attempting to modify the software, be aware of the following:

- BATMAN & ROBIN is currently under development. Modifications and enhancements are made continually. Changes made by others will not be supported by

the original developers. These include alterations to (a) the direct-manipulation human-computer interface, (b) the format and contents of scenario data files generated by ROBIN and executed by BATMAN, (c) the algorithms, code, FSAs, transition rules, and databases involved in platform behavior, (d) the database editing tools, (e) the windowing system's structural design, (f) the code and databases used for displaying maps, and (g) the grid coordinate system.

- BATMAN & ROBIN is a complex system. Tuning the software to run efficiently with different databases and algorithms requires an understanding of several components. Software modifications may appear feasible, but without knowing the limitations of the graphics and computational structures, they may become burdensome or impossible tasks. Many of these issues are beyond the scope of this document and direct consultation with the developers is advisable.

- Contact NPRDC before changing BATMAN & ROBIN. Many modifications may have already been scheduled for a future date, rejected because of limited customer interest, or suspended because of technical problems in the system's design.

5.3 Guidelines for Adding a Computer Model

BATMAN & ROBIN software has been designed and developed as a generalizable, object-oriented, modular system. This readily lends itself to the addition of other computer models to increase the level of fidelity or functionality of BATMAN & ROBIN. To successfully incorporate a new simulation model, some programming guidelines must be followed, specifically:

- The added model must be written in a language compatible with the C programming language and the Sun-4 series computers, such as FORTRAN 77, Pascal, Modula-2, or Ada.

- The new model must consist of functions with well defined software interfaces that are independent of, and compatible with, BATMAN's existing simulation models.

- Any input data the added model requires must be completely separate from the existing BATMAN & ROBIN databases.

- Any output produced by the new model must be initialized and generated independently from BATMAN's output.

- The computer model must not require any user interaction, and it must operate only on data from files or parameters passed to it from BATMAN.

- For the mouse to track smoothly, it must be serviced once every 15 milliseconds. To accommodate this requirement, the model's calculations must be sectioned into reasonable pieces so it can return to BATMAN within this time interval.

- If the simulation model requires functional changes to the existing BATMAN software, these changes must be coordinated with and performed by NPRDC.

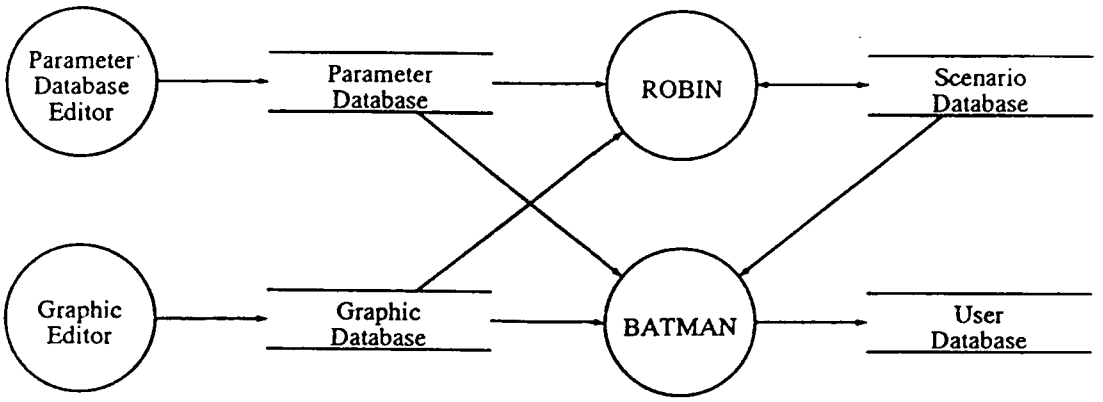


Figure 1. BATMAN & ROBIN Software Components

It is recommended that the added computer model's interfaces or hooks in BATMAN & ROBIN software be written by NPRDC. Since NPRDC is intimately familiar with the software, these interfaces can be implemented efficiently and effectively. Moreover, it eliminates the need for the modeler to have extensive knowledge of the internals of BATMAN & ROBIN.

Because the complexity of simulation models vary dramatically, NPRDC cannot be responsible for the performance of BATMAN & ROBIN after these models have been incorporated. The current version of BATMAN & ROBIN has been highly optimized for speed. Adding a compute-intensive model will likely degrade BATMAN's performance.

6.0 Software Components

BATMAN & ROBIN consists of four databases: Parameter, Graphics, Scenarios, and Users; and three processes: BATMAN & ROBIN, a Parameter-Database Editor, and Graphics Editors. This software composition is illustrated in Figure 1.

The Parameter-Database Editor is used to create and modify the attributes of objects in the Parameter Database, e.g., the fuel-consumption rates of air platforms. Additionally, important platform parameters can be adjusted on a scenario-by-scenario basis using BATMAN & ROBIN's Graphical Frontend to the Database (GFED) (see Section 8.10, **Database and Graphical Frontend Packages**). The GFED is helpful in exploring "what if" scenarios.

To date, the Parameter Database is stored as an ASCII text file. Any editor that creates strict ASCII

files can be used to modify the Parameter database. The developers have used the **vi** editor, a standard Unix utility (SunOS Reference Manual, 1990). BATMAN & ROBIN retrieves values from this database using a hybrid **ndbm** relational database model (see Section 11.0, **Parameter Database**).

The Graphic Editor is used to create and modify BATMAN & ROBIN's graphics or icons. The graphics supported by BATMAN & ROBIN are Sun standard rasterfiles (Pixrect Reference Manual, 1990). In general, all icons and graphic objects should be monochrome since BATMAN & ROBIN apply their own colors to these objects. The developers have used commercial Sun graphics editors, e.g., *IslandPaint*, and a custom software program, *pain44*, developed by NPRDC to create icons and graphic objects. None of these programs are discussed in this document.

7.0 BATMAN & ROBIN Global Data Structures

Table 1 lists global data structures that are used throughout BATMAN & ROBIN software, and identifies the header file where each data structure is defined.

The *SCENARIO_HEAD* data structure, accessed through *sh->*, contains both static global data and miscellaneous scenario-specific data. The static global data is initialized at the start of BATMAN & ROBIN and is available during its entire run. An example of static global data is *sh->debug_on*, which is a boolean value indicating whether BATMAN & ROBIN is being run in debug mode.

Table 1

Data Structure to File Mapping

Data Structure	File	Contents
SCENARIO_HEAD	**scenario.h**	scenario and global data
THEATER	**scenario.h**	scenario log
CONSOLE_FORCE	**force.h**	blue-force resources
PATH_FORCE	**force.h**	path-force resources
STATUS_REC	**force.h**	group or sub resources
TYPE_REC	**force.h**	platform type information
PLATFORM	**platform.h**	one platform
PROJECTILE	**platform.h**	one projectile (weapon)
DETECT_NODE	**platform.h**	one detection-unit (sensor)
BLUE_TEMP	**init_con.h**	blue-force template
PATH_NODE	**init_path.h**	path-force swarm information
PATH_VECTOR	**init_path.h**	path-force flight information
BASE_PORT	**init_path.h**	base/port information

The *SCENARIO_HEAD* also contains a pointer to the *THEATER* data structure, which contains a log of all scenarios currently defined. For more information on the *THEATER* structure, see Section 8.1.10, **Scen_db_access**.

To start BATMAN, the *sh->scen_num* field of the *SCENARIO_HEAD* is set to the appropriate scenario number. The *CONSOLE_FORCE* data structure (Figure 2) and the *PATH_FORCE* data structures (Figure 3) are then built. Access to these structures is gained through the *SCENARIO_HEAD*: *sh->cfh*, *sh->red_pfh*, and *sh->grn_pfh* for Blue-, Red-, and Green-forces, respectively.

The *CONSOLE_FORCE* contains all Blue-force tactical resources organized into one to three task forces: Task Force Alpha (TFA), Bravo (TFB), and Charlie (TFC). Each task force, which can be centered around an aircraft carrier, battle ship, or land base, is divided into *group* and *sub* platforms. Information specific to the *group* and *sub* categorizations is contained in *STATUS_REC*s. Each *STATUS_REC* contains a list of *TYPE_REC*s for each different type of platform in the group. Attached to each *TYPE_REC* is a list of *PLATFORM*s of that type. There is one *PLATFORM* structure for each platform in the simulation. For example, if TFA had four F-14 aircraft, there would be one *TYPE_REC* for the F-14 and four *PLATFORM*s, one for each F-14. Included with each console-force *PLATFORM* are the weapons (*PROJECTILE*s) and sensors (*DETECT_NODE*s) that the *PLATFORM* carries.

One of the plattorms in the *group* category must be a *guide* or *home-base* platform, e.g., an aircraft carrier. This document uses three terms to refer to generic platform categories: *guide*, *mother*, and *sub*. A *guide* platform is the primary one in a task force, capable of carrying *sub* platforms and of being preset in ROBIN. A *guide* platform may be a large or small aircraft carrier, a land base, or something else, e.g., a Tarawa class ship. A *mother* platform is capable of carrying *sub* platforms. For example, a *mother* platform might be a Kidd class ship. Note that a *guide* platform is a *mother*, but that a *mother* is not necessarily the task force's *guide*. A *sub* platform might be an aircraft, e.g., F-14, or a weapon, e.g., Phoenix missile.

BATMAN builds two *PATH_FORCE* data structures: one for the Red force and one for the Green force. Each *PATH_FORCE* has a list of *TYPE_REC*s for each different type of platform in the force, e.g., a Badger aircraft. Attached to each *TYPE_REC* is a list of *PLATFORM*s of that type. There is one *PLATFORM* structure for each platform in the force. For example, if the Red-force had four Badger aircraft, there would be one *TYPE_REC* for the Badger and four *PLATFORM*s, one for each Badger. Attached to each path-force *PLATFORM* is information describing how the platform moves in BATMAN, weapons (*PROJECTILE*s), and sensors (*DETECT_NODE*s).

For ROBIN, only skeletal versions of the *CONSOLE_FORCE* and *PATH_FORCE* data structures are built. These contain just the pertinent information that ROBIN needs, e.g., database parameters from a platform's *TYPE_REC*. Additionally, a different set of data structures is built and maintained during ROBIN. These include *BLUE_TEMP*, *PATH_NODE*, *PATH_VECTOR*, and *BASE_PORT* data structures.

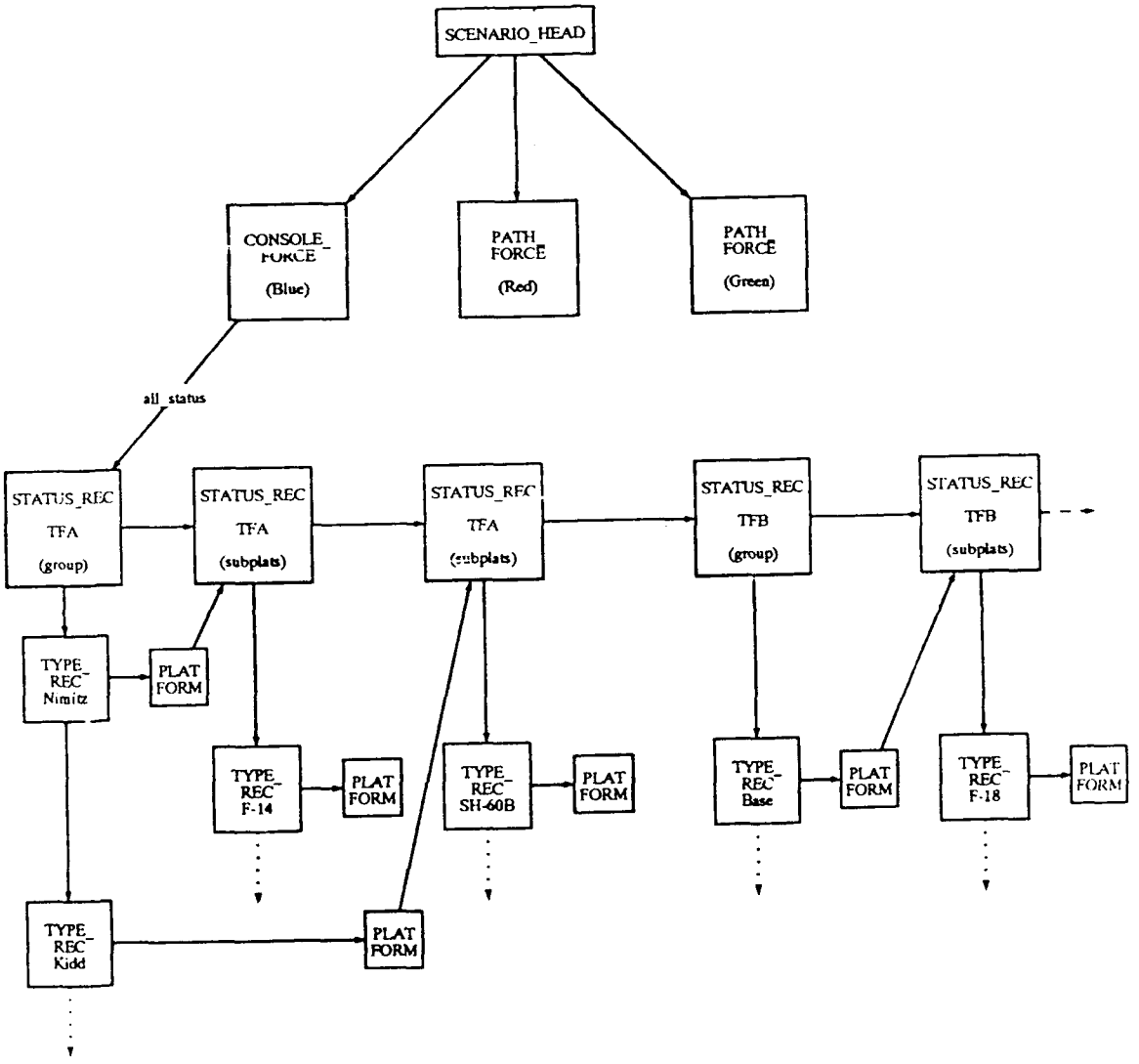


Figure 2. BATMAN CONSOLE_FORCE Data Structure

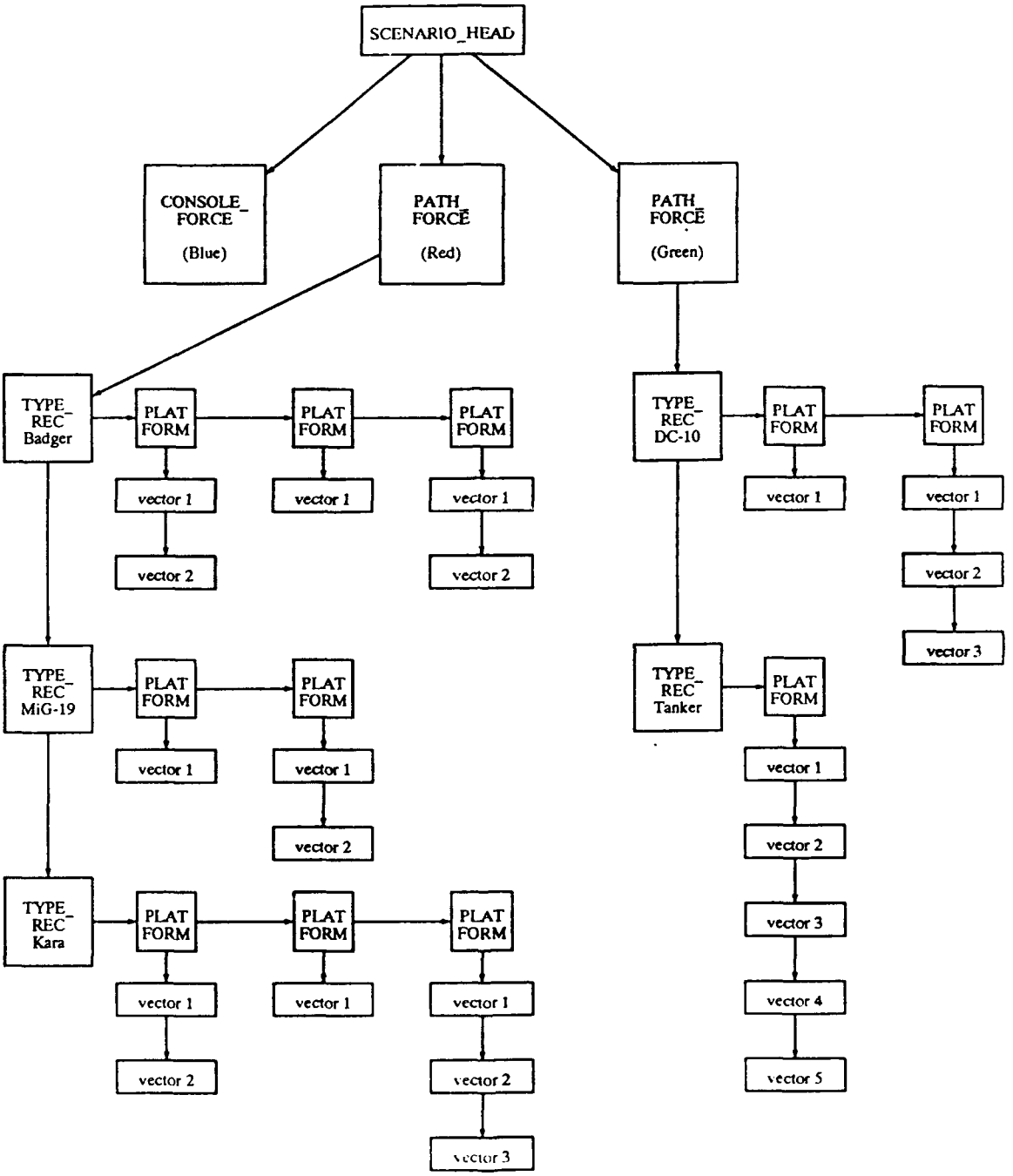


Figure 3. BATMAN PATH_FORCE Data Structures

As illustrated in Figure 4, *BLUE_TEMP* structures are used to build a template of a scenario's Blue-force. Since ROBIN only needs to know what Blue-force resources are in a scenario, the *BLUE_TEMP* scaffold makes for an efficient implementation.

Figure 5 illustrates the data-structure representation of the Red- and Green-forces during ROBIN, with a sketch of how it would appear to the user. The package **robin_vectors** (see Section 8.9.10) builds, maintains, and manipulates this path-force data structure. ROBIN uses *PATH_NODE*s and *PATH_VECTOR*s to build a forest of trees, where each tree represents a swarm of platforms. All of the trees are threaded together by *PATH_NODE next* pointers to build the forest. Attached to each *PATH_NODE* are a list of *PATH_VECTOR*s describing how the swarm moves along that section of the path. When the user breaks apart a swarm, it creates a new level of *PATH_NODE*s for that swarm's tree (see the Badger swarm from Figure 5).

Each path must initiate from a base (air platforms) or port (surface and subsurface platforms). Bases and ports are implemented in the **bases_and_ports** package (see Section 8.11.1) using *BASE_PORT* data structures. A *BASE_PORT* contains an identification number and the latitude-longitude coordinate of the base or port.

8.0 BATMAN & ROBIN Software Packages

The following sections provide an overview of the BATMAN & ROBIN software. For purposes of this description, the software packages are organized into the following groups: (a) Initialization and Control, (b) Loadout and Vector-Logic Grid, (c) Deployment, (d) BATMAN Simulation, (e) Smart Platforms, (f) EW/ESM, (g) ASW, (h) Performance Measures, (i) ROBIN, (j) Database and Graphical Frontend, and (k) Utility. Within each of the following sections, the software packages are described in alphabetical order.

8.1 Initialization and Control Packages

The packages in this group setup the window environment, control the execution flow of BATMAN & ROBIN, initialize scenario data structures, and provide an interface to the user database.

8.1.1 Event_ctrl

This package provides an interposed event function and other miscellaneous event-related functions. The interposed event function (SunView Programmer's Guide, Chapter 17 - Notifier, 1990) provides capabilities to monitor all events as they occur and react to certain events in non-standard ways depending on the application. Hence, it provides the programmer with some flexibility in the way events are handled. BATMAN & ROBIN use this interposition feature to screen for special interrupt events, e.g., ctrl-p, which pauses the simulation.

8.1.2 Init

This package contains functions that perform once-only start-up initialization and clean-up for program exit.

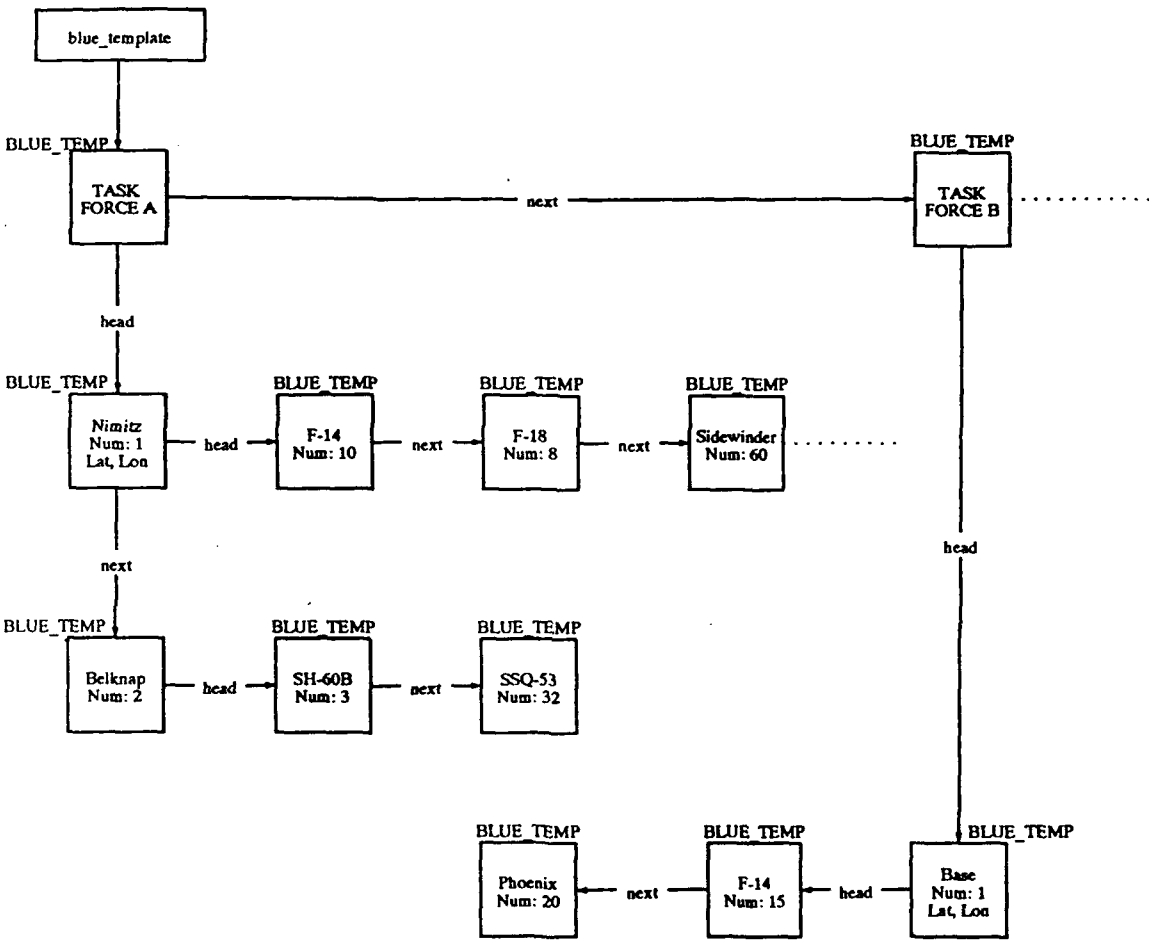


Figure 4. ROBIN Blue-Force Template

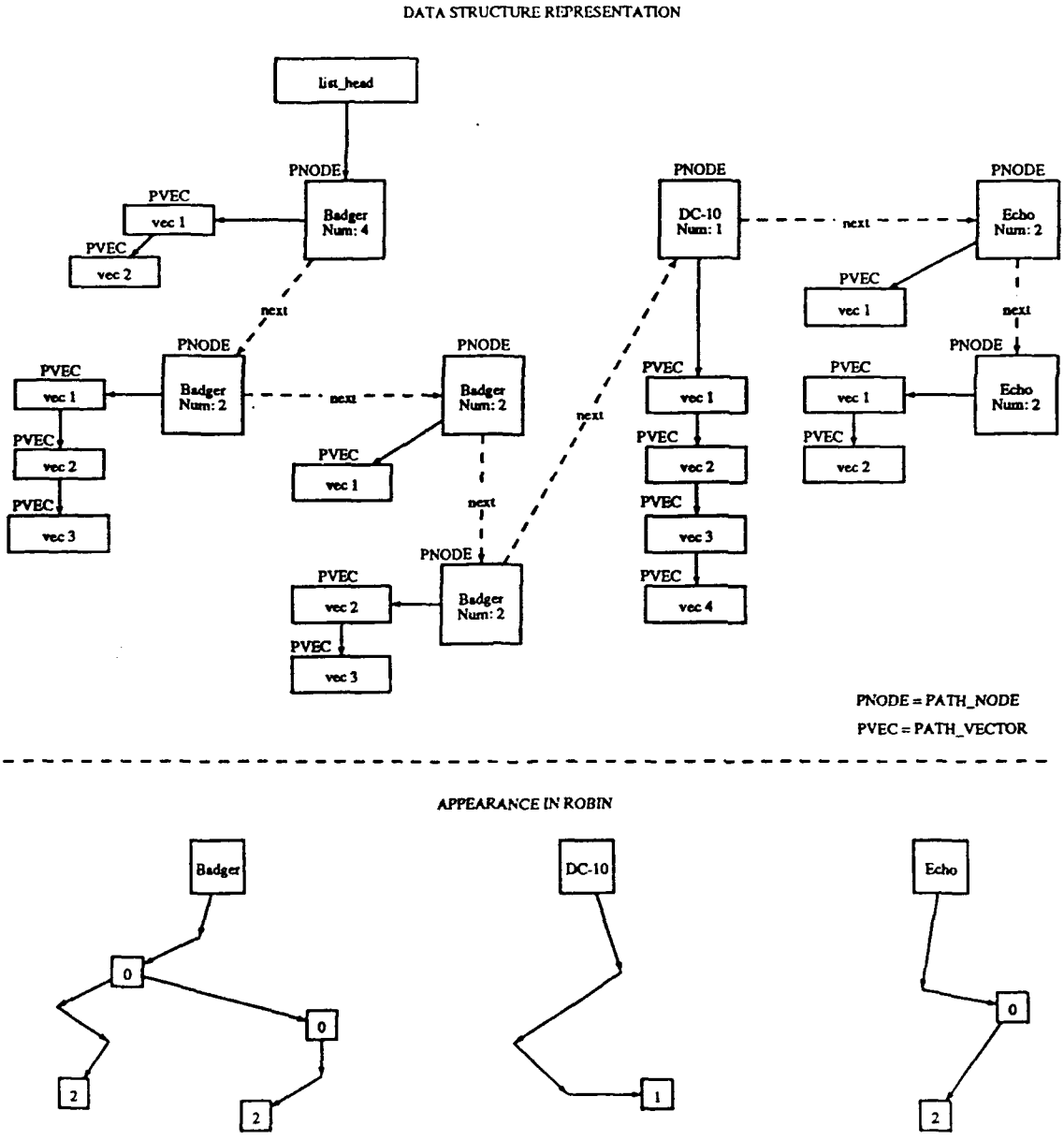


Figure 5. ROBIN Path-Force Data Structures

8.1.3 Init_con

This package initializes BATMAN & ROBIN's Console force data structures and panels. The scenario number must be known before this package can be called, and is used to retrieve the appropriate scenario from the Scenario Database. This package reads in the scenario which specifies the types and numbers of platforms. Then, it allocates records and fills them in with information obtained from the Parameter Database. This package will produce one *CONSOLE_FORCE* data structure, including all *STATUS_RECs*, *TYPE_RECs*, *PLATFORMs*, *PROJECTILEs* and *DETECT_NODEs* that belong to the task force (see Section 7.0, **BATMAN & ROBIN Global Data Structures**).

8.1.4 Init_path

This package initializes the Path force data structures, i.e., Red and Green forces. The scenario number must be known before this package can be called, and is used to retrieve the appropriate scenario from the Scenario Database. Also, it will build a list of flight-path information for each platform. Therefore, this package will produce one *PATH_FORCE* data structure, including all *TYPE_RECs* and *PLATFORMs* that belong to the raiding force (see Section 7.0, **BATMAN & ROBIN Global Data Structures**).

8.1.5 Jtids_antenna

This package creates and destroys the antenna lists attached to the *TYPE_RECs* of JTIDS-capable platforms. Since every *PLATFORM* has a pointer back to its *TYPE_REC*, all platforms have access to their antenna list, which represents the antennas that are available for each type of JTIDS-capable platform, as designated in the Parameter Database (see Part III of this document). For more information, see Section 9.2, **JTIDS**.

8.1.6 Main

This package contains the *main* function and is the primary control package for BATMAN & ROBIN. This package also contains the declaration of the global SCENARIO_HEAD structure, i.e., *sh*.

After the *main* function completes BATMAN & ROBIN initialization, it enters a *while* loop that dispatches window events. Additionally, this while loop processes any programmer defined *loop* functions (*loop* here is a reference to the while loop). *Loop* functions are used, among other things, to control the BATMAN simulation and the ROBIN viewer. For example, the routine *simulation_engine* is added as a *loop* function to drive the BATMAN simulation. Refer to the package **misc** below for more information.

8.1.7 Misc

This package contains functions that are global utilities used by other BATMAN & ROBIN software packages. Most importantly, this package contains the routines *set_loop_func*, *insert_loop_func*, and *remove_loop_func* which add and remove *loop* functions from BATMAN & ROBIN. Refer to the package **main** above for more information.

8.1.8 Playback

This package implements BATMAN & ROBIN's record and playback facility. Playback is a

feature that records user's actions during a BATMAN scenario, and then programmatically recreates the individual's actions during the management of the battle. This is accomplished by recording all user events that occur during allocating, deploying, and managing tactical assets, then regenerating those actions during playback.

8.1.9 Readobj

This package contains routines that will read platform, weapon, and sensor data from the Parameter Database. Both the **init_con** (see Section 8.1.3) and **init_path** (see Section 8.1.4) packages use these routines to build the *CONSOLE_FORCE* and *PATH_FORCE* data structures.

8.1.10 Scen_db_access

This package provides routines to build the *THEATER* data structure and to maintain an index of scenarios from the Scenario Database. These routines are used primarily by ROBIN. For more information, see Section 12.0, **Scenario Database**, in Part III of this document.

One *THEATER* record for each warfare-theater is subsumed under the the *SCENARIO_HEAD*, and contains a log of all scenarios in that area. To date, warfare-theaters are, but not limited to, the following areas: Caribbean, Japan-sea, Bering-sea, Kamchatka-peninsula, South-east-asia, Arabian-sea, Persian-gulf, Mediterranean, North-atlantic, and Murmansk

8.1.11 Timer

This package provides a set-up and a simulation timer. The set-up timer tracks the amount of time the user takes to loadout and deploy Blue forces. The simulation timer tracks the amount of time the user is engaged in battle. This package also contains routines to control pausing and unpausing the simulation.

8.1.12 User_db_access

This package provides routines to interface with the Users Database. For more information, see Section 14.0, **User Database**, in Part III of this document.

8.1.13 User_funcs

This package controls the human-computer interface when the user is performing one of the following functions:

- Listing all users in the User Database.
- Adding a user to the User Database.
- Deleting a user from the User Database.
- Selecting a user to run BATMAN & ROBIN.
- Selecting to run BATMAN & ROBIN in "demo" mode.

This package contains Panel create and P el notify functions to coordinate the above activities. Access to these features is gained through the routine *user_funcs*, exported by this package. *user_funcs* will display a log of all users, and a Panel with the buttons: "Add User", "Delete User",

"Demo", and "Exit".

8.2 Loadout and Vector-Logic Grid Packages

This group of packages creates the tactical-situation screen, handles weapons and antenna loadout for Blue-force air platforms, and implements the vector-logic grid feature.

8.2.1 Grid

This package is responsible for allowing the user to define the range of the vector-logic or defensive grid, and for updating the map and grid when the user chooses to zoom in and out. The specifications for the grid are as follows:

- The simulation plane is assumed to be a Cartesian coordinate system with Victor/Lima (V/L), the defended point, at the origin, (0,0). The grid is drawn with its center at V/L.
- The vector-logic grid is displayed for a full 360 degrees.
- A fixed-range increment for grid labels is used, i.e., 50 nautical mile tic marks.
- Angle or azimuth increments are expressed in integers, e.g., 15 degrees.

8.2.2 Jtids_antenna_load

This package allows users to select antennas for JTIDS-capable Blue-air platforms, and to specify patterns for chosen antennas. JTIDS-antenna selection is accessed from the JTIDS button on the Blue Force Loadout Panel.

If the system allows the specification or selection of a particular antenna, then the relevant aircraft icon depicts the location of all possible antennas that can be activated. When a location is selected, the system presents a set of antenna patterns. Once an antenna is selected, it is activated on the platform, and the system depicts the chosen pattern at the particular location. The antenna can be deactivated by reselecting the location. If an antenna cannot be specified by the user, then the system displays the default antenna location(s). Table 2 lists the current default specifications of antennas for each JTIDS-capable aircraft. For more information on JTIDS, see Section 9.2, **JTIDS**.

8.2.3 Loadout

This package contains functions for creating and interacting with Blue loadout panels which provide the human-computer interface for placing weapons and antennas on Blue air platforms. These panels are created by traversing the list of *TYPE_REC*s attached to the force's *sub* platform *STATUS_REC*. The Blue loadout panel includes:

- An icon that transitions to JTIDS antenna loadout.
- A large icon of the *mother* platform with small icons for the *sub* platforms and weapons that are on it.
- A variable panel-choice item with icons of all the possible weapons that can be loaded out on either of the two *sub* platforms currently displayed.

Table 2

Default Antenna Specifications for JTIDS-Capable Aircraft

Platform	Antenna	Receive	Transmit
E-2C	Left	Omni	Omni[a]
	Right	Omni	Omni[a]
EA-6B	Left	Omni	Omni[a]
	Right	Omni	Omni[a]
F-14	Top	Omni[a]	180°
	Bottom	Omni[a]	---
F/A-18	Top	Omni[a]	180°
	Bottom	Omni[a]	---
S-3B	Top	Omni[a]	180°
	Bottom	Omni[a]	---

[a]User cannot change this antenna's default specification during BATMAN at this time.

8.2.4 Loadout_map

This package creates the tactical-situation screen. The scenario contains all of the information necessary to construct this screen. At this stage of development, the tactical-situation display contains the following:

- A map of the warfare-theater.
- The location of Red- and Green-force bases and ports.
- An icon for each Blue task-force.
- Blue-force DEFCONs.
- A messages icon (if there are any messages for the user to read).
- A Grid icon that, when selected, will leave loadout and advance to grid definition.
- A Deployment icon that, when selected, will bypass grid definition and advance directly to deployment.

8.2.5 Loadout_tf

This package contains functions for creating and interacting with Blue task-force display panels which contain large icons for the force's *home-base* or *guide* and *mother* platforms. Refer to Section 7.0, **BATMAN & ROBIN Global Data Structures**, for a description of *guide*, *mother*, and *sub* platforms. Task-force display panels are created by traversing the list of *TYPE_REC*s attached to the force's *group* platform *STATUS_REC*.

When a *mother* platform on a task-force display panel is selected, the user is "going aboard" the mother with the intention of loading weapons and/or JTIDS antennas onto the task-force's air platforms. When this happens, control is passed to the **loadout** package (see Section 8.2.3).

8.3 Deployment Packages

The packages in this group coordinate deploying Blue task forces in the warfare theater, defining the JTIDS network, and setting initial alert levels for *sub* platforms.

8.3.1 Alert

This package contains routines to model alert or readiness states on *guide* and *mother* platforms. Three alert states are available: Alert 5, Alert 15, and Alert 30. The minutes to launch time are not simulated.

An *ALERT_HEAD* data structure is created and maintained for every *mother* platform in the Console force. The types of platforms in the Alert *PANEL_WIN* are duplicates of those found in the respective *sub* Launch *PANEL_WIN*. Platforms are moved among the readiness states with the function *find_plat_on_level*, which returns the first *sub* platform at the specified alert, and *put_plat_on_level*, which updates the *sub* platform's Alert state. Only *sub* platforms on Ready 5 can be launched. Platforms must be moved from lower to higher alert states for launching, e.g., moved from Ready 15 to Ready 5.

8.3.2 Cf_panels_create

This package creates the Console-force *PANEL_WIN*s, including:

- The Symbol *PANEL_WIN* which contains items for viewing the tactical-situation screen, zooming in or out, moving platforms, erasing platforms. It also includes the *JTIDS deployment functions*: network definition, circuit definition, and relay specification.
- The Task-Forces *PANEL_WIN* which contains an icon for each Blue task force and the Red force hammer-and-sickle icon.
- The Simulation *PANEL_WIN*, which is displayed flush to the right side of the computer screen during BATMAN, and includes the simulation clock, NTDS (Navy Tactical Data System) icons for making specified radar and sonar coverage of Blue force visible, and the icons for displaying the status windows of Blue air, surface, or subsurface platforms.
- *Sibling* Launch *PANEL_WIN*s, one for each Blue task force, containing an icon for each *group* platform in the task force and an icon that returns to the Task-Forces

PANEL_WIN.

- *Sub* Launch *PANEL_WIN*s, one for each task force, containing an icon for each *sub* platform in the task force, an alert icon, a chainsaw icon, a CAP station icon, and an icon that returns to the Task-Forces *PANEL_WIN.*

- Alert *PANEL_WIN*s, one for each task force, containing a Ready 5, 15, and 30 icon for each *sub* platform in the task force. For more information, see Section 8.3.1, **Alert.**

While deploying tactical assets, all of the above but the Simulation *PANEL_WIN* are accessible. During the simulation, all of the above but the Symbol *PANEL_WIN*, the Task-Forces *PANEL_WIN*, and the *sibling* Launch *PANEL_WIN*s are accessible. Therefore, the *sub* Launch *PANEL_WIN*s and the *alert PANEL_WIN*s are used in both deploying assets and managing the battle.

8.3.3 Cf_panels_notify

This package contains the Notify routines for the Console-force *PANEL_WIN*s created by the **cf_panels_create** package (see Section 8.3.2).

8.3.4 Find_symbols

This package contains utilities for locating objects in the *maincanvas* given a mouse-selected coordinate. They are used for move operations, erase operations, and for the range and bearing feature (Section 8.12.5, **Range_and_bearing**). These utilities are used during the deployment and battle-management phases of the simulation.

8.3.5 Jtids_network

This package contains functions for defining a network of JTIDS fighter-to-fighter, air control, and/or surveillance circuits. This feature is accessed through the JTIDS button placed at the top of the Symbol *PANEL_WIN.*

Defining the JTIDS network involves specifying the following:

- *Network Capacity*: percentage of network's capacity allocated to each of the three circuit types. The human-computer interface for this is a Multiple-Partition Slider item (see **Mps**).

- *Circuits*: clusters or sets of JTIDS-capable platforms, i.e., fighter-to-fighter, air control, or surveillance. There can be an unlimited number of circuits. Definition of a circuit is controlled by the routine *circuit_input_handler*, which allows the user to select the platforms that are members of the circuit currently being defined.

- *Relays*: platforms within a particular circuit that can relay JTIDS messages or link data to other platforms in the circuit. As a default, all JTIDS-capable platforms are relays. This feature, implemented by the routine *relay_input_handler*, allows the user to turn a specific platform's relay on or off by selecting it with the mouse on the battle-management display.

For more information, see Section 9.2, **JTIDS.**

8.3.6 Launch_panel

This package contains routines to create, manipulate, and destroy *sub* platform launch *PANEL_WIN* attached to *mother* platforms.

8.3.7 Symbol_manager

This package manages the placement, movement, and removal of objects in the *maincanvas* (where the maps are drawn) during deployment. These objects include Console-force platform, CAP station, and chainsaw icons. The functions in this package are generally called when a mouse selection is made in the *maincanvas*, e.g., specifying the location to deploy an F-14. These calls have to occur within the right context. For example, the F-14 choice in the *sub* launch *PANEL_WIN* is selected prior to deploying it in the *maincanvas*.

8.4 BATMAN Simulation Packages

The packages described in this section control and perform the BATMAN simulation.

8.4.1 Coverage

This package contains radar, sonar, and jam coverage drawing functions used during the simulation as well as functions to manipulate BATMAN & ROBIN's colormap. Manipulating the colormap is particularly useful when switching between sonar and radar coverage.

8.4.2 Detect

This package contains functions that compute positional relationships between platforms and, using the detection-units aboard each of them, determines which platforms a particular sensor can detect. Currently, the Willard-Lueker space-tree detection-model is used to perform these calculations (see Appendix A, **AN O(D+(N $\log_2$ N)) ALGORITHM FOR RANGE/BEARING RESTRICTED SEARCH IN TWO DIMENSIONS**). BATMAN's radar model assumes that a platform being picked-up is within the altitude, azimuth, range, and line-of-sight of the detecting unit, and has a sufficiently large cross section for this sensor.

8.4.3 Engine

This package contains the data structures and code that drives the BATMAN simulation. The controlling function is *simulation_engine*, which operates from a circular list of *ENGINE_NODE*s, as illustrated in Figure 6.

Currently, *ENGINE_NODE*s are grouped into the following types: *clock, map_copy, pre_detect, red_detect_build, red_detect_plat, red_asw_detect, red_post_detect, red_update_plat, green_detect_build, green_detect_plat, green_asw_detect, green_post_detect, green_update_plat, blue_detect_build, blue_detect_plat, blue_asw_detect, blue_post_detect, blue_update_plat, red_launch_plat, green_launch_plat, go_draw, draw_plat, update_esm, display_changes,* and *update_misc*. Figure 6 abstracts these into nine stages: (a) clock, (b) map-copy, (c) red detect and update, (d) green detect and update, (e) blue detect and update, (f) red and green launch, (g) draw platforms, (h) update ESM, and (i) update misc. The list is maintained so that all nodes of the same engine-node-type are contiguous.

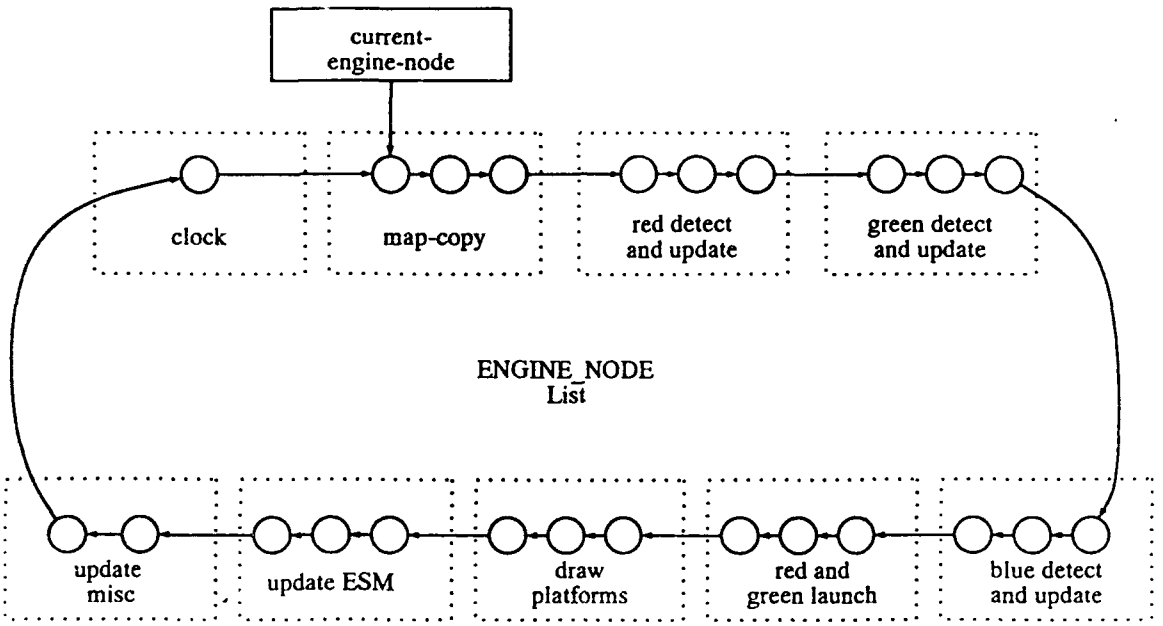


Figure 6. ENGINE_NODE List

Before starting the battle simulation, the *ENGINE_NODE* list is initialized, and the "current-engine-node" is set to map-copy. The initial state of the list is approximate to the illustration in Figure 6. The simulation is started by posing *simulation_engine* as a *loop* function (see Sections 8.1.6, **Main**, and 8.1.7, **Misc**). Then, *simulation_engine* refers to its circular list of *ENGINE_NODE*s to determine which functions it should call. This continues until the simulation is terminated.

Each call to *simulation_engine* is only allowed to run for the amount of time specified by the **msec_engine_update_interval** parameter of the Parameter Database (see Section 11.9, **System-Configuration Parameters**). Imposing this time limit will periodically return control to the system so that it can process user input. Therefore, it may take several calls of *simulation_engine* before one cycle of the *ENGINE_NODE* list is completed. Furthermore, the more platforms involved in the scenario, the longer each cycle will take to complete.

The most powerful feature of the engine is that the *ENGINE_NODE* list it operates from can change during the simulation, dramatically increasing the engine's flexibility. The *ENGINE_NODE* list changes depending on what the battle manager does. For example, if the

battle manager requests that the outer-air battle not be visible, this is handled by removing the *ENGINE_NODE* that draws air platforms. If the battle manager requests for the outer-air to become visible again, the node is simply added back to the list.

8.4.4 Jtids_conn

This package contains functions to coordinate JTIDS connectivity features available during the battle-management. With these features, the user can view a depiction of probable contour or point-to-point connectivity for JTIDS-capable platforms. They are accessed through the JTIDS simulation panel which is placed flush along the left side of the battle-management screen. This package includes functions to (a) create the JTIDS simulation panel, (b) handle notifier events for each feature in the panel, and (c) process mouse input in the *maincanvas* during the JTIDS mode. For more information, refer to Section 9.2, **JTIDS.**

8.4.5 Jtids_hooks

For information about this package, see Section 9.2, **JTIDS.**

8.4.6 Plat_comm

This package communicates detections among all the members of a force, and assigns detections to each platform in a force for use by the smart-platforms engine (see Section 8.5, **Smart-Platform Packages**).

8.4.7 Plat_detect_funcs

These functions use a platform's detection unit, e.g., radar or sonar, to locate hostile platforms, and simulate firing weapons at the detected raiders. Detecting threat platforms is accomplished by the *detect_plats* function from the **detect** package described above, and coordinated by routines in this package. Firing weapons at the detected platforms is also initiated by routines in this package.

8.4.8 Plat_draw_funcs

This package contains functions to draw platforms, CAP stations, chainsaws, radar and sonar coverage, chaff corridors, jamming, antiship missiles, firing lines, and explosions in the *maincanvas* during the simulation. This package can be viewed as a toolbox of drawing functions that update the user's view of the battle. Different drawing functions may be called depending on what happens during the battle simulation. For example, an explosion is drawn when a platform is destroyed. Or, radar coverage for air platforms becomes visible if the user requests it.

8.4.9 Plat_list_funcs

This package contains routines to update a number of lists maintained during the simulation, e.g., a list of destroyed Blue platforms.

8.4.10 Plat_update_funcs

This package contains routines that control and update the movement and status of all Blue, Red, and Green platforms during the simulation. This includes routines called by user input, e.g., request to land an air platform on a carrier, and by the simulation engine, e.g., update a platform's movement on a chainsaw.

8.4.11 Status

This package contains functions to construct and display individual, *group*, and *sub* platform status windows. The *STATUS_WIN* structure, defined in this package, is used to implement these three types of status windows. A *STATUS_WIN* is composed primarily of a Sun *Pixrect* (Sun Pixrect Reference Manual, 1990), but contains additional information to provide more functionality for BATMAN & ROBIN.

One *STATUS_WIN*, declared in the *CONSOLE_FORCE* structure (see Section 7.0, **BATMAN & ROBIN Global Data Structures**), is used to display the status of an individual platform. Only a single platform's status can be displayed at any one time. The individual status window is currently displayed in the lower left corner of the battle display.

Group platform status windows provide the status of all *mother* platforms in the designated task force. *Sub* platform status windows provide the status of all *sub* platforms in the designated task force. A *STATUS_WIN* structure is declared in each *STATUS_REC* from the *CONSOLE_FORCE* structure to hold these windows. The *group* and *sub* platform status windows are currently displayed in the upper left corner of the battle screen.

The display of these three status windows is mutually exclusive, i.e., only one of them can be displayed at a time. When the user requests to view one of these status windows, an "update-misc" *ENGINE_NODE* is added to the *ENGINE_NODE* list to display the selected status window (see Section 8.4.3, **Engine**). When the user no longer chooses to view the status window, the *ENGINE_NODE* is removed from the list.

Throughout BATMAN, Blue platform status is updated as follows: Each *PLATFORM* data structure has a pointer to the force's individual *STATUS_WIN* and the *group* or *sub* platform *STATUS_WIN* to which it belongs. Then, during the battle, each active platform writes its updated status into both the individual *STATUS_WIN* and the appropriate "multiple-platform" *STATUS_WIN* (either *group* or *sub*). The updates are only visible if a status window is displayed.

8.5 Smart-Platforms Packages

The smart-platforms packages implement artificially-intelligent platform behavior, as described in Section 3.0, **Artificially Intelligent or Smart Platform Behavior**. The design can be most easily understood by beginning with the data structures. There are four main tables, all of which reside in **sp_tables**. They are:

> ***SP_RULE_TABLE***: defines all the transition rules in terms of condition functions (from **sp_cond_funcs**), or logical operations upon other rules.
>
> ***SP_STATE_TABLE***: defines all the states in terms of action functions (from **sp_action_funcs**), default speed and area of interest (for detections).
>
> ***SP_FSA_TABLE***: defines all the FSAs in terms of states and rules.
>
> ***SP_MISSION_TABLE***: associates all missions with FSAs and enemy types.

In every cycle of BATMAN's *simulation_engine* (see Section 8.4.3, **Engine**), *sp_engine* is called for all active platforms from "detect and update" stages of the simulation. *sp_engine* checks all

transitions from the current state to see if any can be traversed. Transitions are checked by recursively evaluating the rule which is associated with them. State changes are made until the platform enters a state with no transitions that evaluate to true. If a state change is made, *sp_engine* is responsible for updating not only the current platform, but also any old or new target platforms.

Some rules need to be evaluated for the platform's entire detection list. This is noted in the rule by the *detect_list_eval* field. In this case, the rule is tested for all detections until it evaluates to true. The *target_plat* field in the platform record is used to store the current detection.

User input is handled by *change_plat_direction* in **plat_update_funcs** as was previously the case, but now this function is responsible for instructing *sp_engine* to change state when requested by the user. This is done with *force_new_state* in the **sp_engine** package.

Another noteworthy data structure is the available tankers list which is maintained by **sp_utils**. This list is used as a convenient way to look for nearby tankers when a fighter needs to refuel. It is important that this list be updated every time a tanker is launched or lands.

Distributing detections among members of a force is handled by *esp_detect_plat*. *esp_detect_plat* uses a Willard-Lueker space tree to attach a sorted list of detections that lie within a platform's area of interest to each platform in the force. This list is sorted according to hostile action, and the detection's kill priority and distance from the platform. This list is sorted so that when *sp_engine* picks the first target from the list for which a transition evaluates true, it will be the most appropriate available target.

Because the rules check for the number of attackers before a platform can select a target, the order in which *sp_engine* evaluates the platforms in a force determines how targets will be assigned. To make sure that platforms choose appropriate targets, *esp_detect_plat* sorts the *moving_plats* list for the force so that platforms that are closest to their most likely potential target are allowed to choose a target first.

8.5.1 Sp_action_funcs

This package contains the action functions which correspond to platform states. They control platform movement and weapons firing.

8.5.2 Sp_browser

The smart platforms browser allows the user to examine the FSA diagrams, state descriptions, and transition rules that control artificially intelligent behavior. This information is available during ROBIN and during the simulation in BATMAN and accessed via the "Missions" button. The user traverses this information by first choosing the force (Red, Blue, or Green) of interest (*browser_force_panel_win*) and then choosing a mission (*browser_missions_panel_win*). The *browser_fsa_panel_win* displays the FSA diagram for the mission. The user examines specific rule and state descriptions by clicking on them in the FSA diagram. The selected information is retrieved from a text file and displayed in a SunView text window (*desc_window*).

The FSA diagrams are Sun raster files in **/nprdc/wargame/pics/fsas**, and the state and rule descriptions are in text files in **/nprdc/wargame/data**. There is a separate file for each state description with an extension of **.state** and the rules are all contained in the file **rules.txt**.

8.5.3 Sp_cond_funcs

This package contains the condition functions which are used to evaluate the rules of the platform state diagrams. They check levels of detection, status and attributes of platforms, warnings and weapons status, and user overrides.

8.5.4 Sp_engine

This package contains the smart-platforms engine. This includes functions to evaluate rules, traverse transitions, and changes to new states upon user input.

8.5.5 Sp_tables

This package contains the tables which define the smart-platform rules, states, FSAs, and missions. The tables are all constant, i.e., they are set automatically upon initialization and never modified.

8.5.6 Sp_utils

This package contains smart-platforms utilities. These include routines to manage: memory allocation, an available tankers list, destination location for smart-platforms action functions, smart-platforms platform attributes, display routines, and special detection routines.

8.6 EW/ESM Packages

The purpose of this enhancement to BATMAN & ROBIN is to permit platforms into, or out of, EMCON (emissions control), and to add ESM as a means of detection. The former was accomplished by allowing the scenario creator to select Red or Green platform paths and indicate where radar will be on, and allowing the BATMAN user to turn radar on or off for task forces and/ or individual Blue platforms. The latter was achieved in the following manner. The modeling of radar was augmented to permit platforms with ESM capabilities to detect the individual radars on emitting platforms. These passive detections are displayed on the screen, and used in the smart-platforms decision process. A simulated pop-up status board was added which displays both current and past ESM information. Also added was the option to view past ESM contacts in a depicted warfare theater.

The following assumptions were made during the design of the EW/ESM simulation or model:

- Passive ESM- or counter-detection distances vary as a function of emitter radar type, and range from 1.5 nm to 3.5 nm.
- The emitting and ESM-detecting platforms must be within line-of-sight of one another. This is usually less than the counter-detection range.
- The passively detecting ESM-platform must (a) be within the radar beam or coverage of the emitting platform, and (b) have the capability to detect emitter bands.
- Blue-force platforms know the positions of all other Blue-force platforms, and do not use ESM to passively detect one another.
- ESM passively detects individual radar units emitting from other platforms.
- The location of an emitting radar can be determined only if two or more platforms

passively detect it.

- A platform can be identified as hostile by the radars it emits, if passively detected by ESM.
- An ESM detection is never considered a positive identification of a specific platform type. This requires visual identification.
- It is possible for hostile emitting platforms to "trick" or deceive ESM detection by closely spacing themselves.
- The radar coverage display in BATMAN depicts the maximum possible painted picture.
- If a platform is not under EMCON, all of its search radars will be emitting.
- Search and navigation radars are always pointed in the same direction as the platform's heading, unless it is an omnidirectional sensor.
- All ESM detections -- other than "Don Kay," "Don-2," "Palm Frond", and Green air radars -- are considered hostile.
- Entries are removed from the ESM-status board when they have traveled distances specified by **esm_status_remove_dist** as defined in the Parameter database.
- Entries are removed from the past ESM-contacts display when they have traveled distances specified by **esm_lost_contacts_remove_dist** as defined in the Parameter database.

It was also assumed that Red platforms can be identified when the following radars are passively detected on them:

Udaloy: Top Mesh or Strut Pair
Sovremeny: Top Plate
Kashin: Head Net C and Big Net
Kirov: Top Steer and Big Net
Kiev: Top Pair and Top Sail or Top Sail and Top Steer
Mig-23: High Lark
Mig-25: Fox Fire
Mig-29: NO-93
Mig-31: Foxhound
Su-15: Twin Scan
Mig-19: Scan Odd
Bear: Big Bulge
Mainstay: Suawcs
Badger: Puff Ball
Blackjack: Blackjack
Blinder: Down Beat and Short Horn

ESM detection information is left in the *PLATFORM* records until it has been processed by **esm_display**. **esm_display** is the primary EW/ESM package, controlling the ESM status window in addition to analyzing the detections for the map display.

8.6.1 Esm_detect

This package contains the ESM detection functions, and functions to handle target-acquisition radar.

8.6.2 Esm_display

This package contains the past ESM contacts notify routine, the initialization and clean-up of EW, and the functions that process the ESM detection information and control the ESM display. The display includes ESM detect lines, target acquisition radar, and past ESM contacts. The display routines are responsible for avoiding overlapping detection labels and past contacts, minimizing the detection lines shown, making cross detections briefly appear as single detections, and displaying briefly radar names over recently identified platforms.

8.6.3 Esm_status

This package contains the functions that implement the ESM contacts status window. This includes the notify routine and engine node.

8.7 ASW Packages

BATMAN & ROBIN's ASW interfaces contain the following features:

- Loading sonobuoys and laying them in specific patterns from SH-2, SH-3, SH-60, S-3B, and P-3C ASW aircraft.
- Towed and hull-mounted passive sonars for surface and subsurface platforms.
- Dipping active sonar for SH-3s and SH-60s.
- Active and passive sonar-detection models.
- Range determination by any two or more cross-bearings between passive sonar-detection lines.
- Launch and recovery of ASW helicopters from Blue-force surface platforms.

The following sections describe the software packages that implement the human-computer interfaces for incorporating ASW capabilities in BATMAN & ROBIN.

8.7.1 Asw_pattern_generator

This package contains functions that generate a list of latitude-longitude coordinates describing how an ASW aircraft should fly while laying specific sonobuoy patterns. To calculate the coordinates, *create_asw_drop_list* is called with the name of a sonobuoy pattern, a series of mouse clicks that indicate the pattern's location and orientation, and the number of sonobuoys to be laid. The number of mouse clicks required to define the pattern depends on the pattern configuration or

type, and will be either two, three, or variable. The spacing between sonobuoys also depends on the pattern type, and will either be 1.5 MDR (median-detection range), 2/3 or 1 CZ (convergence zone), or, in the case of the Freelance pattern, exactly as the user specifies. Table 3 lists the available sonobuoy patterns and the ASW aircraft that can lay each pattern.

The Chevron, Circle, and Semi-Circle patterns are treated differently than the others in that their size will be calculated automatically to accommodate the number of sonobuoys the user wishes to drop. For example, dropping ten sonobuoys along a Circle pattern using MDR spacing will yield a larger circle than if five sonobuoys were dropped.

Table 3

Available Sonobuoy Patterns

Pattern	ASW Platforms
Barrier	SH-2F, SH-3F, SH-60B
Chevron	SH-2F, SH-3F, SH-60B
Circle	SH-2F, SH-3F, SH-60B
Semi-Circle	SH-2F, SH-3F, SH-60B
Freelance	SH-2F, SH-3F, SH-60B
Dipping Sonar	SH-3F, SH-60B
5-6-5 MDR	S-3B, P-3C
5-6-5 CZ	S-3B, P-3C
Brushtac	S-3B, P-3C
Barrier	S-3B, P-3C
Sawtooth Barrier	S-3B, P-3C
4x4	P-3C

8.7.2 Asw_sonar

This package contains functions that implement simulated ASW, and can be be divided into two groups:

1. functions that support the laying of sonobuoy patterns, and
2. functions that interface with the active and passive sonar-detection models.

The human-computer interface for laying sonobuoy patterns allows users to drop specified or selected patterns from ASW air platforms. The user specifies the number, longevity, depth, radio frequency, pattern, and orientation of the sonobuoys. The **asw_pattern_generator** package (see Section 8.7.1) is used to compute the path vectors that an air platform flies when laying a particular

pattern.

The remaining functions provide links with the active and passive sonar-detection models (see Section 9.3, **Anti-Submarine Warfare**). To date, the following active or passive sonars are available: hull-mounted, towed arrays or tails, and dipping.

8.7.3 Lambda_sigma

For information about this package, see Section 9.3, **Anti-Submarine Warfare.**

8.8 Performance Measures Packages

The packages in this group compute and display objective, automatically recorded, multivariate performance measures (sometimes referred to as statistics) for the most recent gaming of a specific BATMAN scenario. Performance measures can be viewed at the console, sent to a printer, or written to a file. The header file **stats_recs.h** declares four data structures used by the performance measures packages. Their names and purposes are listed in Table 4.

Table 4

Performance Measures Data Structures

Data Structure	Purpose
STATS_MANAGER_WIN	Performance Measures Manager *PANEL_WIN*
STATISTICS_WIN	Performance Measures *PANEL_WIN*
STATS_FIELDS	Hold data collected during simulation
STATS_HEAD	Ties above three structures together

STATS_MANAGER_WIN provides a *PANEL_WIN* that allows the user to select the types of platforms (air, surface, or subsurface) and point of view (Red or Blue) of the performance measures display. *STATISTICS_WIN* provides a *PANEL_WIN* to hold the performance measures when they are displayed. One of its Panel Items is a large *Pixrect* where the performance measures are written. *STATS_FIELDS* holds the data collected during the simulated battle that will be used to compute the performance measures. *STATS_HEAD* brings the above three structures together in one place, with some other information global to the performance measures packages.

8.8.1 Stats

This package contains functions that initialize the *STATS_HEAD* structure, create the *PANEL_WIN*s in the *STATS_MANAGER_WIN* and *STATISTICS_WIN* structures, and display or print performance measures.

8.8.2 Stats_compute_funcs

The functions in this package use the data collected during the simulated battle to compute the performance measures. The *STATS_FIELDS* structure holds the collected data. These functions can be used to calculate the performance measures from either the Red or Blue Force's point of view. In general, one function is dedicated to calculating each performance measure.

8.8.3 Stats_notify

This package contains the Notify functions for the *PANEL_WIN*s from the *STATS_MANAGER_WIN* and *STATISTICS_WIN* structures.

8.8.4 Stats_update_funcs

This package contains functions that the simulation uses to send battle-related data to the *STATS_FIELDS* structure, thereby recording a history of the battle as it occurs. Many of the functions from the simulation engine make periodic calls to these functions.

8.8.5 Stats_verify

This package contains functions that write performance measures to a file in the user's directory. For more information, see Section 14.0, **User Database.**

8.9 ROBIN Packages

The packages in this group implement ROBIN, the system's rapid scenario generator. They are responsible for providing a friendly interface to creating, editing, and viewing Red-force raids for different warfare theaters as well as assigning tactical assets to Blue task forces and defining the Green or neutral-force movements.

Figure 7 provides a map between the features provided by ROBIN and the primary package providing that feature. The header file **robin_globals.h** contains global variables used by all ROBIN packages.

8.9.1 Robin_assign

This package provides the routines for assigning specified sets of scenarios to particular groups of students or system users. That is, scenarios or test items can be clustered into sets or tests, and users or students are placed into groups or classes. Within a set, scenarios can be administered either randomly or in a specified order to a group of BATMAN users.

8.9.2 Robin_blue

This package contains the Panel creation and notify routines that allow the user to specify the Blue force's tactical assets, and the routines that build and maintain the Blue-force template (see Figure 4 in Section 7.0, **BATMAN & ROBIN Global Data Structures**). The user can define up to three task forces designated TFA, TFB, and TFC. For each task force, the user indicates the air, surface, and subsurface platforms, weapons, and sonobuoys for the scenario. In BATMAN, these will become the Blue-force tactical assets that the battle manager has to allocate, deploy, and control. **Robin_blue** also allows the user to change important database parameters of blue-force platforms (see **Database and Graphical-Frontend Packages** below).

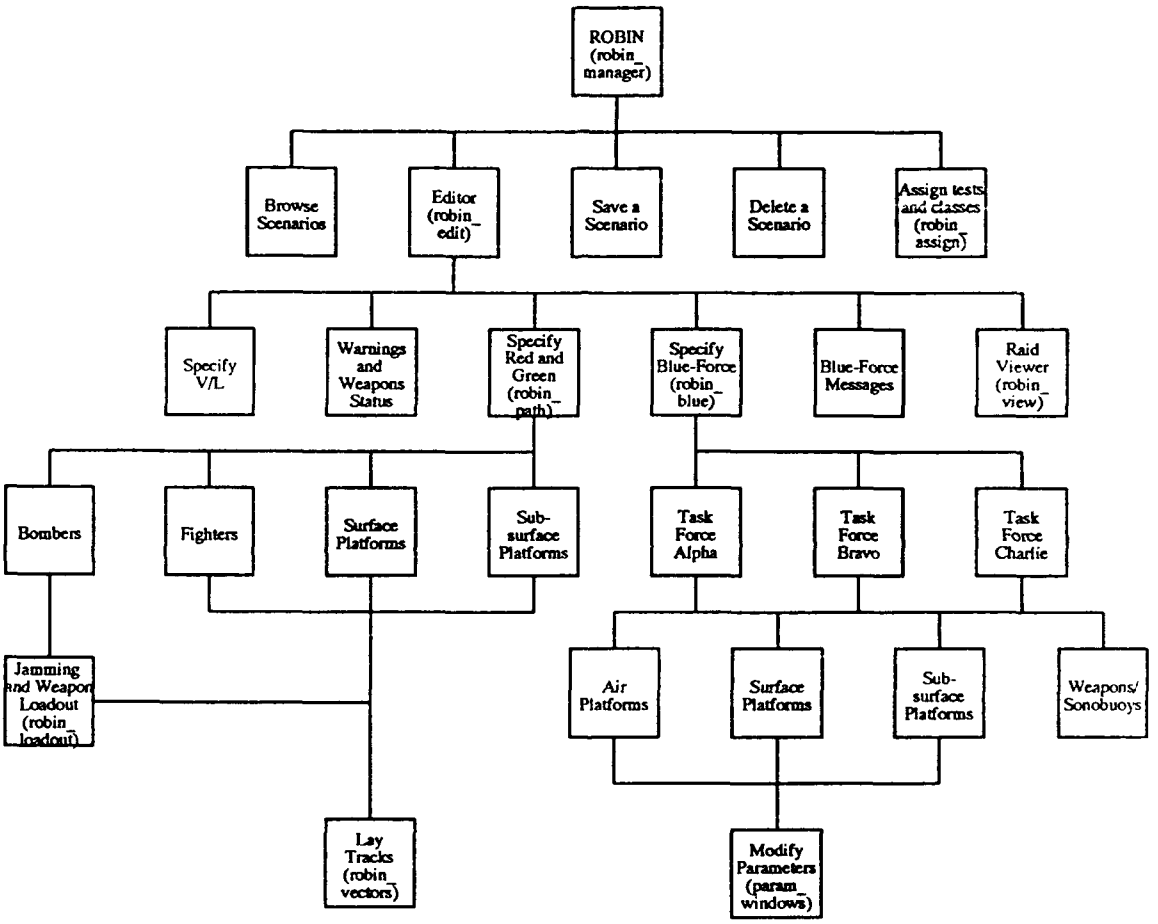


Figure 7. ROBIN Features to Package Map

8.9.3 Robin_edit

This package provides controlling routines for the scenario editor. It creates the editor *PANEL_WIN* and regulates transitioning to specific edit features such as placing the defended point (V/L), specifying the Red-force raid, or allocating Blue-force assets. Only one scenario may be edited at a time.

8.9.4 Robin_init

This package contains functions that initialize ROBIN's global variables, control the creation of ROBIN's windows and panels prior to a session, and remove ROBIN's windows and panels after a session.

8.9.5 Robin_io

This package contains functions for writing and reading scenarios to and from the Scenario Database. For more information on the Scenario Database, refer to Part III of this document.

8.9.6 Robin_loadout

This package contains routines for loading Red-force bombers with anti-ship missiles as well as communications-jamming and radar-jamming pods. For more information on jamming, refer to Section 9.2, **JTIDS**.

When the user selects a bomber to specify its flight path or track, the Red-bomber loadout screen is displayed. This screen includes:

- A large icon of the bomber and its designation.
- Icons for possible weapons that this bomber can carry.
- A communications-jamming pod icon.
- A radar-jamming pod icon.
- An icon for transitioning from loadout to flight paths.

Weapon icons allow the user to loadout weapons on the selected bomber, the same way that weapons are loaded onto Blue platforms (see Section 8.2.3, **Loadout**).

The communications-jamming and radar-jamming pod icons are both toggle items. Initially, these pods are displayed unloaded. When selected, each is "loaded" onto the bomber (indicated by reverse video), and its associated jamming attributes are displayed. If selected again, the pod is unloaded, and the jamming attributes are cleared from the screen. Jamming-pod attributes are defaulted if they are not specifically set.

8.9.7 Robin_manage

This package controls ROBIN's administrative features including listing scenarios, saving scenarios to disk, deleting scenarios, copying scenarios, and transitioning to the scenario editor. It also contains functions that create the manager *PANEL_WIN*.

8.9.8 Robin_map

This package provides routines that interface with the *World Database II* maps (see Section 9.1, **World Database II**). Specific features provided by this package include displaying warfare theaters, interacting with the miniature world map that appears in the upper left corner of the ROBIN manager, drawing the scale that borders the warfare theaters, and placing and drawing V/L within a warfare theater.

8.9.9 Robin_path

This package contains the Panel creation and notify routines that permit the user to specify the Red-force attack and the Green-force movement paths or tracks. These routines are generic so that they can handle either Red or Green forces, i.e., the force serves as a parameter to many of these routines. In fact, these routines could implement other hypothetical forces, e.g., Yellow. The data structures *PF_INFO* and *PF_DATA* store path-force specific information. This package is tightly coupled with the **robin_vectors** package.

8.9.10 Robin_vectors

This package contains routines for creating and maintaining the path-force "forest" data structure (see Figure 5 in Section 7.0, **BATMAN & ROBIN Global Data Structures**). This package also contains routines for placing Red- and Green force tracks including laying a new track, extending an existing track, erasing defined tracks, hiding tracks from view, turning radar on or off, laying chaff, and/or jamming on tracks. This package is tightly coupled with the **robin_path** package.

8.9.11 Robin_view

This package provides routines that will preview the Red-force raid showing how these hostile platforms will maneuver during BATMAN, where their radars are on, and where they will lay chaff corridors or jam communications. Similar to BATMAN's simulation engine, the routine *set_loop_func* is used to trigger the ROBIN viewer (see Sections 8.1.6, **Main**, and 8.1.7, **Misc**).

8.10 Database and Graphical-Frontend Packages

The packages in this group interface to the Parameter Database and implement the Graphical Frontend (GFED) to the Database. The GFED permits important platform parameters to be adjusted on a scenario-by-scenario basis. For Blue-force platforms, the GFED is accessed in ROBIN during Blue task-force specification for a specific scenario. When a parameter change is made, it is applied to every platform of that task-force. For Red- and Green-force platforms, the GFED is accessed in ROBIN during path specification. When a parameter change is made, it is applied to every platform in the path for that specific scenario.

8.10.1 Database

This package contains functions for retrieving values from BATMAN & ROBIN's Parameter Database. The design of this package is a hybrid **ndbm** relational database model (see Section 11.0, **Parameter Database**).

The function *init_database* reads the Parameter Database into a memory-resident hash table (see Section 8.11.6, **Hash**). *d_get_i* retrieves a parameter with an integer value from the Parameter Database hash table. *d_get_s*, *v_get_s*, and *d_get_data* all retrieve parameters with character string

values. *traverse_database* searches the Parameter Database hash table for a specified key, and executes a function on all matching elements.

8.10.2 Param_attribute_manager

This packages handles all the properties specific to individual parameter types, such as their names, location in the *PLATFORM* record, and data type.

8.10.3 Param_data_manager

This package contains routines to manipulate the data structures used by the GFED to modify database parameters. It also contains routines for reading and writing to the scenario database (i.e., **blue.n** and **path.n**), as well routines to access the parameter database files.

8.10.4 Param_windows

This package contains all the window creation, deletion, and notify routines for the GFED. *init_parameters_panel* initializes GFED's panel which include a centered item to display the platform's icon. *show_parameters_panel* invokes the GFED by displaying GFED's panel with the specified platform icon and a list of changeable parameters for the platform.

8.11 Utility Packages

The packages in this group provide miscellaneous utilities used throughout BATMAN & ROBIN.

8.11.1 Bases_and_ports

This package implements Red- and Green- force *bases and ports*. Bases and ports are treated differently in ROBIN than they are in BATMAN, because paths are treated differently in ROBIN than they are in BATMAN (see Section 7.0, **BATMAN & ROBIN Global Data Structures**).

In ROBIN, *PATH_NODE*s are used to define Red- and Green-force paths. Therefore, bases and ports threads the *PATH_NODE*s to determine which paths initiate from each base or port.

BATMAN does not use *PATH_NODE*s to define Red- and Green-force paths. Therefore, since BATMAN only needs to know what resources originate from any base or port, a list of *PLAT_PAIR*s is used for BATMAN.

8.11.2 Canvas_win

This package provides routines for creating and manipulating *CANVAS_WIN*s. A *CANVAS_WIN* is based upon a SunView Canvas, but contains additional information to provide more functionality. A powerful feature of this package is that it provides a mechanism whereby a Canvas can contain pseudo *Panel-button* items.

8.11.3 Chaff

This package contains functions for drawing and updating chaff depictions during the simulation. Chaff is implemented as a *GRAPHIC_ORGANISM* (see Section 8.11.5, **Graphic_organism**) that develops or expands as the simulation proceeds.

Each of the routines in this package manipulates the chaff *GRAPHIC_ORGANISM* in some

manner: *build_chaff_organism* creates a chaff *GRAPHIC_ORGANISM*; *update_chaff* expands the chaff corridor; *go_chaff_draw* partially or totally redraws the chaff; *go_chaff_reset* resets the chaff in preparation for a total redraw; and *go_chaff_free* deletes the chaff *GRAPHIC_ORGANISM*.

8.11.4 Colors

This package provides functions that initialize and maintain BATMAN & ROBIN's colormap which resides on disk in the file specified by the **colormap** parameter of the Parameter Database (see Section 11.9, **System-Configuration Parameters**). The SunView routines *pr_putcolormap* and *pw_putcolormap* use this colormap to set-up its color palette.

8.11.5 Graphic_organism

*GRAPHIC_ORGANISM*s are graphical entities that develop or expand throughout the simulation, finally reaching a predetermined final appearance, e.g., chaff (see Section 8.11.3, **Chaff**). They can be drawn incrementally as they evolve, or redrawn completely if necessary.

The programmer defines a *GRAPHIC_ORGANISM* with the routine *go_add*, and specifies: (a) a draw function, (b) a free function, (c) a reset function, and (d) a generic data pointer passed to each of these three functions.

A list of all defined *GRAPHIC_ORGANISM*s is kept, and the exported routines *go_draw_all*, *go_reset_all*, and *free_go_list* are used, respectively, to draw, reset, and free the *GRAPHIC_ORGANISM*s.

8.11.6 Hash

This package contains basic hash table functions. Each hash table is implemented as a collection of *(key, data)* pairs, such that the data are retrieved via the key. Among other things, this package is used to store a memory-resident copy of the Parameter Database.

8.11.7 List_manager

This package provides standardized data structures and functions for manipulating doubly-linked lists. Each *LIST_NODE* contains previous and next pointers as well as a pointer to some generic data.

8.11.8 Memory

This package contains functions for handling memory and pixrect allocation. It also contains routines to track and debug memory-allocation bugs.

8.12 Messages

This package contains routines to create and edit messages in ROBIN, and to display and read messages in the ROBIN viewer and BATMAN.

8.12.1 Mps

This package provides the routines and data structures to create and manipulate a Multiple-Partition Slider (MPS). Based upon a Sun-slider item (SunView Programmer's Guide, 1990), an MPS contains *N* partitions in one slider. The sum of all partitions in an MPS equals 100 percent.

An MPS item is created and used like any Sun panel item, e.g., a slider or button. Sun-message items are used to implement the two portions of an MPS: (a) the picture area holds the graphical representation of the partition, and (b) the label area, below the picture area, displays the labels and numerical values of the partitions.

8.12.2 Number_pad

This package provides a pop-up number pad that is used for inputting integers into BATMAN & ROBIN, e.g., specifying the number of Phoenix missiles to load onto each F-14. The number pad is implemented with a *PANEL_WIN* where each digit (0 - 9), the backspace key, the enter key, and the display area are Panel items. If the enter key is selected and the number in the display area is within a specified range, the number is "accepted" and the number pad is automatically cleared from the screen.

8.12.3 Panel_win

This package provides routines for creating and manipulating *PANEL_WIN*s. A *PANEL_WIN* is an object created specifically for BATMAN & ROBIN. It is based on a SunView Panel, but contains additional information to provide more functionality. A powerful feature of this package is that it can create a SunView Panel-choice item with a variable number of options where each is labeled with an icon. For example, this feature is used during weapons loadout. The number of different types of weapons that can be loaded out on an aircraft varies depending on the specific platform. The **panel_win** package can be used to handle these situations without requiring a different Panel-choice item for each air platform.

8.12.4 Popup_panel

This package provides a mechanism for displaying "pop-up" (overlaid) messages, which may contain button and/or a password-request items. It can be used to display general or error messages with a continuation button, "pseudo menus" where each option is a button, or a simple password request.

One private *PANEL_WIN* is used to store and display the popup messages. Upon each request to display a message, the local *PANEL_WIN* is cleared and then set-up with the specified attributes of the message. The "varargs" parameter passing mechanism is used to specify a varying attribute list that defines the appearance of the message panel. Defaults are used when required attributes are not specified.

8.12.5 Range_and_bearing

This package contains functions that implement the range-and-bearing feature available during both deployment and the simulation. When in range-and-bearing mode, the user may obtain the range-and-bearing between any two points on the map display or screen. If the user clicks near a platform or sonobuoy, the location of the object is used for calculating the range and bearing rather than the location of the click. The range-and-bearing information appears in a box near the midpoint of the line segment selected. This information remains on the screen until the user selects another two points or until the user exits range-and-bearing mode. When range-and-bearing is selected during the simulation, it pauses the simulation.

The range-and-bearing feature works by replacing the current *maincanvas* input handler with

rb_input_handler, which controls the selection of the two endpoints and updates the rubberband line. The user can exit range-and-bearing by clicking on the range-and-bearing icon or by selecting one of the other active icons on the screen. The function *clear_range_bearing* uses the values of local static variables to restore the previous state of the system. This restoration includes resetting the *maincanvas* input handler to its previous value, removing any range-and-bearing information from the screen, and unpausing the simulation if necessary.

8.12.6 Scen_display

This package creates a Panel that displays a graphical summary of a scenario including the warfare area, scenario number, and the Red and Blue-force tactical assets. It is used in ROBIN by the manager and in BATMAN by playback and the performance-measures packages.

8.12.7 Time_item

This packages creates and manipulates a standard time-item implemented as four buttons. Selecting any of the buttons will bring up the number-pad. The caller specifies the time-item's initial value, the number-pad's location, and the number-pad's enter function in the call to *create_time_item*.

8.12.8 Ud_rop

This package contains user-defined raster operations (UD-ROPs). The only UD-ROP to date, *clear_radar*, clears the radar depiction from the display. This package is intended to contain special-purpose raster operations that cannot be performed by the standard *PIX_** macros (Pixrect Reference Manual, 1990).

8.12.9 Utilities

This package contains utility routines for trigonometric calculations, memory allocation, image drawing, and several other miscellaneous functions. These functions are used throughout BATMAN & ROBIN.

8.12.10 Version

This package contains a string indentifying the current version of BATMAN & ROBIN. This is the string that is displayed in the upper-left-hand-corner of BATMAN & ROBIN's frame.

8.12.11 Warnings_and_wpn_status

This package contains the Blue- and Red-force warnings and weapons status code. It also contains code to construct the Blue-force DEFCON-warning boxes displayed on the TACTSIT screen.

8.12.12 Zoom

This package handles the zoom feature available in ROBIN while defining or editing a scenario and in BATMAN while deploying or managing tactical assets. When activated, it provides a special mouse cursor and input handler to inform the user that the zoom feature is engaged. When a new range is selected, the **mapdb** package is called to reset and draw the new view (see Section 9.1, **World Database II**).

9.0 Software Interfaces

This section describes BATMAN & ROBIN's current software interfaces for integrating external computer models. These interfaces, or software hooks, provide a means for linking other models with BATMAN & ROBIN. For specific details and caveats regarding model integration, refer to Section 5.3, **Guidelines for Adding a Computer Model.**

9.1 World Database II

The warfare theaters in BATMAN & ROBIN are based upon the *World Database II* maps that were developed and provided by the Applied Physics Laboratory of Johns Hopkins University. This document does not describe the map database or the software that controls it, only BATMAN & ROBIN's interface to it.

The specifications of these maps are as follows:

- The maps are rendered by an orthogonal projection of the globe onto a plane tangent from a specified latitude-longitude coordinate, and viewed from a specified range. For a given scenario, the center point of the projection is Victor/Lima.
- The resolution of the views range from 16 to 2048 miles in diameter.
- Land masses are drawn with a polygon filling algorithm that draws the largest possible polygon first, and then fills in the details by drawing smaller and smaller polygons.
- The maps can be drawn with or without country borders.
- The maps can be drawn with or without latitude-longitude lines.

In interacting with the *World Database II* maps, BATMAN & ROBIN use the coordinate systems described below and illustrated in Figure 8.

- The Global coordinate system is based on latitude and longitude. In ROBIN, all raid information will be stored in this system as well as the center point (V/L) of the battle area.
- The Map Plane is a Cartesian coordinate system tangent to the globe at the center point of the battle area with Y axis *parallel* to the longitude line of the center point. The map is an orthogonal projection from the globe to this plane. X and Y distances are in miles from the center of the earth along a plane through the center, not along the surface. The tangent point of this plane will not change during the simulation.
- The Simulation Plane is a Cartesian coordinate system with origin at the center of the battle area. X and Y are in increments of .1 miles. The distance in miles is linear on this coordinate system, which is used for detection and calculation of all platform movements during the simulation. Platform locations will be stored in this coordinate system during the simulation, and will be converted to the other systems when needed.

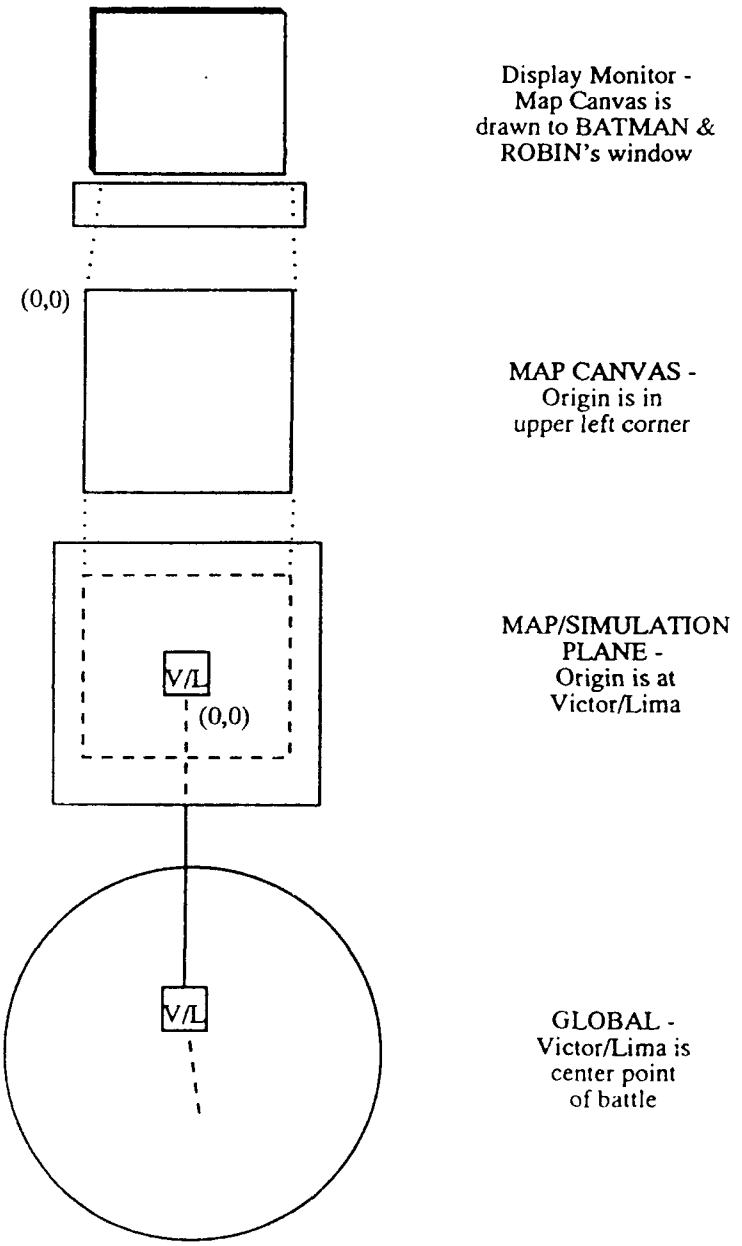


Figure 8. BATMAN & ROBIN Coordinate Systems

- The Map Canvas is the coordinate system of the *maincanvas* containing the visible portion of the Map Plane. The center of this coordinate system corresponds to an offset from the origin of the Map Plane. This coordinate system is used to draw all objects on the *maincanvas*, i.e., the battle area.

9.1.1 draw_map_on_pr

BATMAN & ROBIN interfaces to the *World Database II* maps through the *draw_map_on_pr* routine in the **mapdb** package located at **/nprdc/wargame/mapdb/mapdb.c**. This routine draws a map onto a *Pixrect* with the specified attributes. Its formal interface is as follows:

```
extern int draw_map_on_pr(lat, lon, x_offset, y_offset, range, borders, lat_lon_lines,
                          land_color, sea_color, lake_color, border_color, latlon_color,
                          pr, star_field_pr, debug_on)

    double lat, lon;                /* latitude-longitude coordinate of V/L */
    double x_offset, y_offset;      /* offset from V/L */
    int range;                      /* range from V/L */
    int borders;                    /* whether country borders should be drawn */
    int lat_lon_lines;              /* whether lat-lon lines should be drawn */
    unsigned int land_color;        /* color of land */
    unsigned int sea_color;         /* color of sea */
    unsigned int lake_color;        /* color of lakes */
    unsigned int border_color;      /* color of country borders */
    unsigned int latlon_color;      /* color of lat-lon lines */
    Pixrect *pr;                    /* pixrect to draw the map onto */
    Pixrect *star_field_pr;         /* background of globe */
    int debug_on;                   /* whether BATMAN & ROBIN is in debug mode */
```

Several other routines exported by **mapdb** convert locations between the coordinate systems described above, e.g., *mc_to_mp* converts map-canvas coordinates to map-plane coordinates. *draw_map_on_pr* initializes data in **mapdb** so that these coordinate transformations are accurate.

9.2 JTIDS

This section describes BATMAN & ROBIN's software interfaces for JTIDS (Joint Tactical Information Distribution System) computer models. There are three JTIDS interfaces: *get_jam_radius*, *jtids_pt_pt_connectivity*, and *jtids_contour_connectivity*; and they all reside in the package **jtids_hooks**.

9.2.1 get_jam_radius

The purpose of *get_jam_radius* is to compute the jamming radius in miles for a bomber loaded with

the specified jamming pod. The integer returned is the jamming radius. Its formal interface is as follows:

```
extern int get_jam_radius(plat, pod_type, uhf, vhf, lx, x, k, ku, ant_pat)
        char *plat;             /* name of bomber */
        int pod_type;           /* 0 = comm-pod; 1 = radar-pod */
        int uhf;                /* pod's UHF percentage (0 - 100) */
        int vhf;                /* pod's VHF percentage (0 - 100) */
        int lx;                 /* pod's Lx percentage (0 - 100) */
        int x;                  /* pod's x percentage (0 - 100) */
        int k;                  /* pod's k percentage (0 - 100) */
        int ku;                 /* pod's Ku percentage (0 - 100) */
        int ant_pat;            /* pod's antenna pattern (45, 90, 180, or 360) */
```

Jamming pods are loaded on Red-force bombers when the scenario is created with ROBIN (see Section 8.9.6, **Robin_loadout**). The scenario creator also specifies where along the bomber's flight path or track jamming should take place.

It is intended for this routine to be called in three places: (a) when communications and radar jamming is assigned to a bomber's flight path (see Section 8.9.10, **Robin_vectors**); (b) during the ROBIN viewer (see Section 8.9.11, **Robin_view**); and (c) during the BATMAN simulation (see Section 8.4.8, **Plat_draw_funcs**).

9.2.2 jtids_pt_pt_connectivity and jtids_contour_connectivity

These routines are called during the BATMAN simulation and their purposes are, respectively, to compute point-to-point and probable contour connectivity for a specified JTIDS-capable platform. They are called by the **jtids_conn** package described in Section 8.4.4.

The formal interface for *jtids_pt_pt_connectivity* is as follows:

```
extern PLATFORM *jtids_pt_pt_connectivity(blue_plats, red_plats, selected_plat, npg)
        PLATFORM *blue_plats;   /* list of all blue platforms */
        PLATFORM *red_plats;    /* list of all red platforms */
        PLATFORM *selected_plat; /* the plat selected by the operator */
        NPG_TYPE npg;           /* network participation group */
```

where:

typedef enum npg_types

{FF_CIRCUIT, AC_CIRCUIT, SURV_CIRCUIT, NPG_TYPES} NPG_TYPE;

jtids_pt_pt_connectivity returns a list of *PLATFORM*s from the *blue_plats* list, identifying who within the specified *npg* the *selected_plat* can communicate with. Note that a *PLATFORM* contains its location on the Simulation Plane (see Section 9.1, **World Database II**).

The formal interface for *jtids_contour_connectivity* is as follows:

```
extern Pixrect *jtids_contour_connectivity(blue_plats, red_plats, selected_plat,
                    tangent_lat, tangent_lon, map_range, resolution)

    PLATFORM *blue_plats;     /* list of all blue platforms */
    PLATFORM *red_plats;      /* list of all red platforms */
    PLATFORM *selected_plat;  /* the plat selected by the operator */
    double tangent_lat;       /* latitude of map tangent point (V/L) */
    double tangent_lon;       /* longitude of map tangent point (V/L) */
    int map_range;            /* current zoom level of the map */
    int resolution;           /* resolution of the contour in tenths of a mile */
```

The returned pixrect describes the probable contour connectivity for the *selected_plat*. The connectivity determination is based on platform locations, antenna patterns, circuit and relay definitions, and Red jamming. Bits, which are set in the pixrect, represent contours or areas where the *selected_plat* can probably or likely communicate.

tangent_lat, *tangent_lon*, *map_range*, and *resolution* are included in the software interface so that transformations can be made between the Simulation Plane and the Global coordinate system (see Section 9.1, **World Database II**). It is assumed that these parameters would be required in rendering the contour pixrect.

9.3 Anti-Submarine Warfare

This section describes BATMAN & ROBIN's software interfaces for ASW sonar-detection models. The sonar models currently used by BATMAN & ROBIN are the passive and active versions of the sonar equations developed for BATMAN & ROBIN (Buoni, 1989).

THESE ARE DESCRIPTIVE, NOT PHYSICAL, SIMULATION MODELS THAT WERE SELECTED TO BE COMPUTATIONALLY EFFICIENT IN ORDER TO MINIMIZE BATMAN & ROBIN'S RESPONSE TIME. In addition to the assumptions mentioned in

Buoni's thesis (1989),, there are two other notable constraints:

- The models operate in two-dimensional space. That is, platform and sensor depths are not factored into the models. Likewise, the effect of depth on sound propagation is not considered.
- The models do not consider frequency. BATMAN & ROBIN assume that all platforms radiate sound in a frequency range detectable by all sensors. Likewise, the effect of frequency on sound propagation is not considered.

The **lambda_sigma** package contains the functions that implement the active and passive sonar-detection models. The sonar-detection process is conducted in two phases. The first phase uses the Willard-Lueker *traverse_and_prox* function (see Section 8.4.2, **Detect**) to single out those target platforms that are within range of the detecting sonar.

The second phase calculates either the active or passive sonar equation for each of the candidate target platforms to determine if they have been detected. The active and passive sonar equations are as follows:

Active:

$$SE = SL - (NL - DI) - 2TL + TS + X(t)$$

Passive:

$$SE = SL - (NL - DI) - TL + X(t)$$

Where:

SE is Signal Excess, or that part of the signal which is audible above the background noise.

SL is Source Level, or the intensity of the signal source. For passive detection, this is the amount of sound generated by the target platform and is a function of the platform's speed. For active detection, this is the amount of sound generated by the active sonar.

NL is Noise Level, or the background noise from which the desired signal must be extracted. This is a combination of ambient noise in the ocean and self noise generated by the platform carrying the sonar device. Self noise is also a function of the platform's speed.

DI is Directivity Index, or a measure of the sonar's ability to detect sound from all directions.

TL is Transmission Loss, or the weakening of the sound as a function of distance from its source.

X(t) is a random term intended to account for the stochastic nature of the detection process. This term is calculated using a Lambda-Sigma Jump process (Buoni, 1989).

TS is Target Strength, or the ability of a target platform to reflect sound. Target Strength is only

used in the active equation.

The differences between the two equations are that:

1. the active model takes into account Target Strength (**TS**),
2. Transmission Loss (**TL**) is doubled in the active model since the sonar signal must travel to the target and then back to the active sensor, and
3. Source Level is an attribute of the detector in the active equation, and an attribute of the target platform in the passive equation.

A detection occurs when Signal Excess (**SE**) becomes greater than the detection unit's Detection Threshold.

BATMAN & ROBIN's software interface to these sonar-detection models is through three functions exported from **lambda_sigma**: *init_random_jump*, *random_jump_time*, and *sonar*. *init_random_jump* and *random_jump_time* control the generation of the **X(t)** term in the sonar equations, and *sonar* is the interface to both the passive and active sonar equations.

9.3.1 init_random_jump

This function initializes the static variables *overall.lambda* and *overall.sigma* in the **lambda_sigma** package. These variables are constants used by the Lambda-Sigma Jump process and control the random number distributions of the process. It is called once before each run of the BATMAN simulation. Its formal interface is as follows:

```
extern void init_random_jump()
```

9.3.2 random_jump_time

This function updates the **X(t)** term in the sonar equations. It is called once for every detection opportunity of each detection unit. A global Lambda-Sigma Jump process contributes half of the **X(t)** term and a Lambda-Sigma Jump process specific to each detection unit contributes the other half. Its formal interface is as follows:

```
void random_jump_time(lambda_sigma)
        JUMP *lambda_sigma;        /* variables for the Lambda-Sigma Jump process */
```

9.3.3 sonar

This function calculates the sonar equation for a specified sonar and target to determine if a detection has occurred. Its formal interface is as follows:

```
extern void sonar(dp, pp, ux, uy)
        DETECT_NODE *dp;        /* the sonar detection unit */
        PLATFORM *pp;           /* the target platform */
        int ux;                 /* distance along x-axis from sonar to target */
        int uy;                 /* distance along y-axis from sonar to target */
```

This function examines a field in **dp* to determine if the sonar is active or passive, and then calculates the appropriate sonar equation, either the active or passive. The *ux* and *uy* parameters are used to determine the distance from the sonar to the target platform. If the sonar successfully detects the target, then the target platform (**pp*) is added to a list that is kept in the sonar (**dp*).

Part IV: Database Descriptions

10.0 Purpose and Scope

This part of the document describes the BATMAN & ROBIN databases. It is intended primarily for software-maintenance personnel. However, those individuals with a technical background who want to understand the organization of the databases may also benefit from reading this part of the documentation. **At this time, the databases described herein are completely unclassified and initially intended for development and evaluation purposes only. These databases which are independent of the simulation software can be made classified if so desired.** The following sections describe the Parameter, Scenario, Graphic, and User Databases.

11.0 Parameter Database

The Parameter Database contains operational attributes of air, surface, and subsurface platforms, sensors, weapons, sonobuoys, and miscellaneous system-configuration parameters. BATMAN & ROBIN was designed and developed to be generic and adaptable. That is, platforms, weapons, or system-configuration parameters may be easily added, deleted, or modified without changing source code. Relevant parameters are stored in a standard text file that can be altered with any text editor, e.g., **vi**. This text file is referred to as the Parameter Database, and can be either classified or unclassified depending upon the user's needs.

11.1 Hybrid Ndbm Relational Database Model

BATMAN & ROBIN retrieves values from the Parameter database using a hybrid **ndbm** relational database model. Several relational and object-oriented database management systems were

evaluated in choosing this hybrid **ndbm** implementation (see Appendix C, **Relational Database Considerations and Issues**). We began using an **ndbm** implementation, but soon realized three significant drawbacks to this package: (a) the total size of a key/data record could not exceed 4,096 bytes; (b) concurrent updating and reading is risky since **ndbm** utilizes no file-locking or reliable cache flushing; and (c) the **ndbm** database would be difficult to modify since it is not stored in ASCII format. To mitigate these deficiencies, we designed a hybrid **ndbm** system that maintains the original objectives stated in Appendix C, **Relational Database Considerations and Issues**.

In our hybrid approach, we wrote database access and maintenance routines in C modeled after **ndbm**. We also maintained the original file syntax of the Parameter database and kept it in ASCII format, easing modifications to the database and facilitating frontending the database with a graphic interface (see Section 8.10, **Database and Graphical-Frontend Packages**).

This database design does not preclude us from using more sophisticated databases in the future. If other groups need to connect our database to ORACLE, INGRESS, SYBASE, or some other commercial database, code could easily be written to extract the necessary values from these databases and store them in the target database.

11.2 Location and Format

The Parameter Database is located in the directory specified by the Unix environment variable *DATABASE* as indicated in the user's **.login** file. If this variable is undefined, it defaults to **/nprdc/wargame/batman/batdb**. The Parameter Database is distributed in the following files:

assignments.db	**asw.db**	**blue_air.db**	**blue_params.db**
blue_ships.db	**blue_subs.db**	**blue_weapons.db**	**browser.db**
colors.db	**ew.db**	**green_air.db**	**green_params.db**
green_ships.db	**green_subs.db**	**jtids.db**	**pictures.db**
radar.db	**red_air.db**	**red_params.db**	**red_ships.db**
red_subs.db	**red_weapons.db**	**stats.db**	**sysparm.db**
user_funcs.db			

The lines in these files consist of an object, parameter, and value, formatted in the following way:

/object/parameter **"value"**

An **object** specifies the name of a platform, weapon, sensor, sonobuoy or some other parameter, e.g., F-14. A **parameter** represents an attribute of an object, e.g., altitude. A **value** assigns a particular number or string to the parameter, e.g., 30,000 feet. The value must be enclosed by quotation marks. For example, the following line would tell BATMAN & ROBIN that F-14s should fly at 30,000 feet:

/F-14/altitude "30000"

The following entries provide further elucidation of how objects are defined in BATMAN &

ROBIN:

```
/F-14/class              "platform"
/F-14/force_id           "blue"
/F-14/stats_type         "air"
/F-14/long_name          "Tomcat"
/F-14/altitude           "30000"
```

These values define the "F-14" as a Blue air platform flying at 30,000 feet, which can be referred to as "Tomcat".

For the purposes of this discussion, the Parameter database is divided into ten sections: (a) Platform, (b) Weapon, (c) Sensor, (d) JTIDS, (e) ASW, (f) Icon, (g) System-Configuration, (h) Performance-Measures, (i) User-Database, and (j) GFED.

11.3 Platform Parameters

alert_pic: the alert icon displayed on a platform's launch panel. This parameter is only relevant for surface platforms and land bases.

> ***value***: the name of the file that contains the icon.

altitude, **altitude_max** and **altitude_min**: a platform's default, upper, and lower bounds in feet above or below sea level.

> ***value***: air platforms greater than zero, surface platforms zero, and subsurface platforms less than zero.

antennas: available antennas for a JTIDS-capable platform.

> ***value***: a list of antennas separated by spaces where each is an **antenna** object (see Section 11.6, **JTIDS Parameters**) defined elsewhere in the Parameter Database, e.g.,

```
/E-2C/antennas                  "E-2Cleft_antenna E-2Cright_antenna"
.
.
.
/E-2Cleft_antenna/name          "LEFT"
/E-2Cleft_antenna/power         "1200"
/E-2Cleft_antenna/transmit_pattern"360"
etc.
```

auto_loadout: whether weapons should be automatically loaded onto the platform by the system at initialization time, rather than manually by the person using BATMAN & ROBIN.

> ***value***: y (yes) or n (no); the default is n.

base_types: the platform types that a platform can launch from and return to. This parameter is

only relevant for air platforms.

value: a list of platforms separated by spaces, e.g.:

/F-14/base_types "Nimitz Base"

caps_and_chains: whether or not the combat-air patrol (CAP) and chainsaw icons should appear on a platform's launch panel. This parameter is only relevant for surface platforms and land bases.

value: y or n.

change_direction_func: the name of the C function called when the user moves a platform.

value: currently, *change_plat_direction* is appropriate for all platforms.

class: indicates an object's type.

value: PLATFORM, WEAPON, RADAR, SONAR, SONOBUOY, or TAIL (i.e., tail sonar).

detection: the detection-units (sensors) associated with a platform.

value: a list of sensors separated by spaces where each is a **sensor** object (see Section 11.5, **Sensor Parameters**) whose parameters are defined elsewhere in the Parameter Database. The following example allocates two sensors to each F/A-18, whose attributes are expected to be defined later in the Parameter Database:

```
/FA-18/detection            "FA-18air_radar FA-18surf_radar"
.
.
.
/FA-18air_radar/range       "100"
/FA-18air_radar/points      "4"
/FA-18air_radar/azimuth     "90"
etc.
```

draw_func: the name of the C function called to draw a platform's icon.

value: *draw_plat_mb_jammed* will not draw a Blue platform when it is jammed; *draw_plat_ao_cj* will not draw a Red platform unless it has been detected; and *draw_missile* will draw Red-force AS missiles.

for_detection: Whether or not the identified sensor for this platform should have its detections processed during BATMAN. This parameter provides a mechanism for ignoring irrelevant sensors to optimize BATMAN, thereby improving its user-response time.

value: y or n; the default is y.

force_id: whether a platform belongs to the Red or Blue force.

value: BLUE or RED.

fuel_cons_launch: This platform's fuel-consumption in pounds during launch.

value: a positive integer varying by platform.

fuel_consumption_full_power: This platform's fuel-consumption rate in pounds-per-hour when traveling at **speed_full_power**.

value: a positive integer varying by platform.

fuel_consumption_launch: This platform's fuel-consumption rate in pounds-per-hour when launching.

value: a positive integer varying by platform.

fuel_consumption_max_conserve: This platform's fuel-consumption rate in pounds-per-hour when traveling at **speed_max_conserve**.

value: a positive integer varying by platform.

fuel_max: the maximum amount of fuel in pounds that an air platform can carry.

value: a positive integer.

hit_tolerance_level: the amount of damage a platform can withstand before being destroyed (see *Appendix* B, **BATTLE-DAMAGE-ASSESSMENT SIMULATION**). This parameter is used in conjunction with a weapon's **damage_pts** to model platform damage (see Section 11.4, **Weapon Parameters**).

value: equal to or greater than one. Air platforms are generally assigned a tolerance level of one; surface and subsurface platforms are assigned greater tolerance levels depending on their displacement.

give_fuel: the amount of *give* fuel in pounds a tanker aircraft, e.g., KA-6D, usually carries.

value: a positive integer.

id_number: a unique number assigned to each type of object that BATMAN & ROBIN uses to identify it.

value: refer to Table 5 for the range of valid ID numbers of different platforms.

keep_out_range: Detections within this keep out range will be evaluated by the platform when it is occupied, e.g., escorting a threat, refueling, moving in for a VID, etc..

value: A positive integer significantly smaller than **surveillance_range**. For optimal BATMAN performance, this parameter should not be larger than necessary.

kill_priority: Priority used to help platforms determine whom to attack.

value: a positive integer greater than or equal to one.

large_picture: the large icon of a platform or weapon.

value: the name of the file that contains the graphic object.

long_name: the NATO, or other, name for a platform, e.g., "BACKFIRE" for the Soviet TU-26.

value: depends on the platform.

Table 5

Object Identification Numbers

Object Type	Id Number
Blue Air	0 - 99
Blue Surface	100 - 199
Blue Subsurface	200 - 299
Blue Weapons	300 - 399
Blue Sonobuoys	400 - 499
Red Air	500 - 599
Red Surface	600 - 699
Red Subsurface	700 - 799
Red Weapons	800 - 899
Green Air	900 - 999
Green Surface	1000 - 1099
Green Subsurface	1100 - 1199

max_attackers: The maximum number of enemy platforms that will close to attack this platform.

value: a positive integer greater than or equal to one.

medium_picture: the medium-size icon of a platform or weapon.

value: the name of the file that contains the graphic object.

min_visible_cross_section: a platform's cross-section used by the detection algorithm to mimic sighting it.

value: a positive integer.

mission: The smart-platform's default mission, i.e., the FSA it executes or follows.

value: to date, the available missions are: BLUE_ATTACK_AIRCRAFT_MISSION,

BLUE_FIGHTER_AIRCRAFT_MISSION, BLUE_AIR_SURVEILLANCE_MISSION, BLUE_AIR_TANKERS_MISSION, BLUE_AIR_ASW_MISSION, BLUE_SHIPS_MISSION, BLUE_SUBMARINES_MISSION, GREEN_AIR_MISSION, GREEN_SHIPS_MISSION, GREEN_SUBMARINES_MISSION, RED_ATTACK_AIRCRAFT_MISSION, RED_AIR_SURVEILLANCE_MISSION, RED_FIGHTER_AIRCRAFT_MISSION, RED_SHIPS_MISSION, RED_SUBMARINES_MISSION, and RED_ANTISHIP_MISSILE_MISSION.

patterns: the sonobuoy patterns that an ASW platform can lay (see Section 8.7.1, **Asw_pattern_generator**). This parameter is only relevant for ASW air platforms, i.e., SH-2, SH-3, SH-60, S-3, and P-3C.

value: a list of sonobuoy patterns separated by spaces, e.g.:

/SH-2F/patterns "barrier chevron circle semi_circle freelance"

prefix: the label displayed below a platform's icon during BATMAN & ROBIN.

value: any string with four or less characters.

radar_cross_section: a platform's radar-cross section used by the detection algorithm.

value: a positive integer.

rotate_resolution: rotate resolution used for making platforms point in the direction that they move.

value: a number between 1 and 360 that indicates the number of possible orientations of the platform icon. A value of 1 indicates that an orientation is possible every single degree, i.e., 360 icons; while, a value of 360 indicates that only one icon orientation is possible for this platform, i.e., one icon pointing vertical.

side_picture: a side-view icon of a platform.

value: the name of the file that contains this graphic object.

small_picture: a small icon of a platform or weapon.

value: the name of the file that contains the graphic object.

self_noise: a list of self-noise levels in decibels produced by a platform for each knot of speed. For example, if the maximum speed of a platform is 30 knots, then the list of self-noise values must include 31 numbers: one for when the platform is not moving, and one for each knot of speed. The first number in the list represents that a platform is not moving, and the last number in the list represents that a platform is moving at maximum speed. This parameter is used by the active and passive sonar-detection models (Buoni, 1989) as a component of Noise Level (NL), and is only relevant for surface and subsurface platforms.

value: a list of positive integers.

sonar: the sonars available to a platform.

value: a list of sonars separated by spaces, e.g.,

/P-3C/sonobuoy_detection/sonar "SSQ-36 SSQ-53 SSQ-62 SSQ-77"

or:

/LosAngeles/tail_detection/sonar "tail"

The detection unit used by a sonar is included as part of the parameter, e.g., "/sonobuoy_detection" in the P-3C example above.

source_level: a list of sound-source levels in decibels produced by a platform for each knot of speed. For example, if the maximum speed of a platform is 30 knots, then the list of source-level values must include 31 numbers, one for when the platform is not moving, and one for each knot of speed. The first number in the list represents that the platform is not moving and the last number in the list represents that the platform is moving at maximum speed. This parameter is used by the passive sonar-detection model (Buoni, 1989), and is only relevant for surface and subsurface platforms.

value: a list of positive integers.

speed_def, **speed_max** and **speed_min**: the default, upper, and lower velocity bounds of a platform or weapon in nautical miles per hour.

value: positive integers.

speed_full_power: This platform's speed when traveling at full-military power.

value: a positive integer varying by platform.

speed_max_conserve: The most fuel-efficient speed for this platform.

value: a positive integer varying by platform.

stats_type: a platform's type used by the performance-measurement utility.

value: AIR, SURF, SUB, or MISSILE.

surveillance_range: Detections within this surveillance range will be evaluated by the platform when it is unoccupied.

value: If this platform's **mission** is BLUE_FIGHTER_CAP_MISSION, then this value should be twice the **fighter_sector_radius**; otherwise, it should be equal to the longest range weapon this platform can fire. For optimal BATMAN performance, this parameter should not be larger than necessary.

tanker_types: tanking platforms that can refuel a particular platform. If it is a tanker, then this lists the aircraft that it can refuel.

value: a list of platforms separated by spaces, e.g.:

/KA-6D/tanker_types "A-6 A-7 F-14 FA-18 A-4 F-4"

target_strength: the target strength in decibels of a platform. The value of this parameter is dependent on the size of the target platform: the larger the target platform, the greater the **target_strength**. This parameter is used by the active sonar-detection model (Buoni, 1989), and is only relevant for surface and subsurface platforms.

value: a positive integer.

visual_range: the approximate distance a platform's pilot or captain can see in nautical miles.

value: a positive integer.

weapons: the types and number of weapons that a platform can carry.

value: a list of weapons separated by spaces where each type is prefixed by a positive integer. If the weapon is not prefixed by an integer, then the number defaults to one. For example, the following allocates two Harpoon missiles and one Rockeye missile to the A-6 aircraft:

/A-6/weapons "2 harpoon rockeye"

11.4 Weapon Parameters

altitude_max and **altitude_min**: the maximum and minimum altitude of the target platform.

value: a positive integer.

damage_pts: the amount of damage a weapon will inflict on its target at impact. This parameter is used in conjunction with a platform's **hit_tolerance_level** to model platform damage (see Section 11.3, **Platform Parameters**).

value: a positive integer.

max_launch_altitude and **min_launch_altitude**: the maximum and minimum altitude of the platform that fires this weapon.

mnemon: the mnemonic name of a weapon which is used to identify it in BATMAN.

value: any string with four or less characters.

prob_kill: the probability of the weapon destroying its target.

value: an integer between 0 and 100.

range: the maximum effective distance in nautical miles of a weapon.

value: a positive integer.

rounds_per_burst: the number of rounds fired in one burst by simulated guns on Blue platforms.

value: a positive integer.

time_for_tar: The time in seconds that this weapon needs its target acquisition radar (TAR) on before it fires.

value: a positive integer varying by weapon.

tar_name: The name of the TAR this weapon uses for fire control.

value: The name of the TAR, e.g., "Down Beat".

weapon_type: The weapon's type.

value: Either PROJECTILE_WEAPON or MISSILE_WEAPON. PROJECTILE_WEAPONs, when they are successful, instantaneously hit their targets and inflict damage. MISSILE_WEAPONs travel to their target at a specified speed and can be detected and intercepted.

11.5 Sensor Parameters

altitude_max and **altitude_min**: a sensor's upper and lower bounds in feet above or below sea level.

value: positive and negative (subsurface) integers.

azimuth: the azimuth of radar or sonar coverage in degrees.

value: a positive integer between 1 and 360.

counter_range: the passive ESM- or counter-detection distance for this radar in nautical miles.

value: a positive integer greater than the emitting radar's range.

detect_func: the name of the C function called to detect platforms.

value: a C function name.

detection_threshold: the detection threshold in decibels used by the active and passive sonar-detection models (Buoni, 1989). The detection threshold is the ratio of signal to noise that constitutes a detection for a particular sonar device. See Urick (1983) for a detailed description of this value.

value: a positive integer.

directivity: the directivity index in decibels used by the active and passive sonar-detection models (Buoni, 1989). The directivity index is a measure of the sensor's ability to detect sound from all directions. Urick (1983) contains a detailed description of the derivation of this value.

value: a positive integer.

effective_altitude_min and **effective_altitude_max**: the minimum and maximum altitudes a sensor can detect. For example, sonar's **effective_altitude_min** might be -100, and its

effective_altitude_max might be -1.

value: an integer.

emit_bands: The ESM bands that this sensor emits on.

value: list of upper-case letters separated by spaces, e.g., "I J".

emitter_name: The name of this sensor.

value: varies by sensor, e.g., "Don Kay".

min_target_cross_section: the minimum cross-section size of a platform that a particular sensor can detect.

value: a positive integer.

points: the number of points in the polygon that depicts radar or sonar coverage.

value: an integer between 4 and 16.

range: a sensor's radar range in nautical miles.

value: a positive integer.

source_level: the source level in decibels used by the active sonar-detection model (Buoni, 1989). The source level is the amount of sound produced by an active sonar device.

value: a positive integer.

11.6 JTIDS Parameters

circuit/ac: the types of platforms allowed in a JTIDS air-control circuit.

value: Currently: E-2C, EA-6B, F-14, F-18, A-18, S-3B, Ticonderoga, and Nimitz.

circuit/ff: the types of platforms allowed in a JTIDS fighter-to-fighter circuit.

value: Currently: F-14 and F-18.

circuit/surv: the types of platforms allowed in a JTIDS surveillance circuit.

value: Currently: E-2C, EA-6B, F-14, F-18, A-18, S-3B, Ticonderoga, and Nimitz.

name: the name of the specific antenna referred to by BATMAN & ROBIN.

value: LEFT, RIGHT, TOP, BOTTOM, or MAIN.

power: the antenna's power in watts.

value: a positive integer.

transmit_pattern: the antenna's default transmit pattern in degrees.

value: 0, 45, 90, 180, or 360.

receive_pattern. the antenna's default receive pattern in degrees.

value: 0, 45, 90, 180, or 360.

pattern_loadout: when present, this parameter identifies an antenna pattern that must be selected.

value: transmit or receive.

hot_spot_x: the x-coordinate mouse location for the particular antenna, relative to (0,0) on the platform icon that carries the antenna (see Section 8.2.2, **Jtids_antenna_load**).

value: calibrated by the programmer for each user-selected antenna.

hot_spot_y: the y-coordinate mouse location for the particular antenna, relative to (0,0) on the platform icon that carries the antenna (see Section 8.2.2, **Jtids_antenna_load**).

value: calibrated by the programmer for each user-selected antenna.

angle: direction the antenna points where 90 degrees points up.

value: 0, 90, 180, or 270 (typically 0 for RIGHT antennas, 90 for TOP antennas, 180 for LEFT antennas, and 270 for BOTTOM antennas).

11.7 ASW Parameters

The ASW parameters are divided into Patterns, Environment, and Sonobuoys.

11.7.1 Patterns

CZ: the distance in nautical miles used to represent convergence zones (CZs) which is used in spacing sonobuoys for some patterns.

value: a positive real number, typically between 28 and 32.

clicks: the number of mouse clicks required by the user to indicate where a sonobuoy pattern should be located (see Section 8.7.1, **Asw_pattern_generator**).

value: 0 for the freelance pattern, 2 for linear patterns (e.g., barrier), and 3 for geometric patterns (e.g., circle).

line_click: used for geometric patterns and indicates when an orientation line should be specified and drawn for the pattern.

value: 2 is appropriate for all geometric patterns.

MDR: the distance in nautical miles used to indicate median-detection-range (MDR) which is sometimes used for sonobuoy spacing.

value: a positive real number, typically 1.5.

picture: the icon used for selecting a specific sonobuoy pattern during BATMAN.

value: the name of the file that contains the icon.

tail_length: the length in nautical miles of towed sonar arrays or tails attached to ships and submarines.

value: a positive integer.

tail_spacing: the spacing between sonars on a towed array or tail.

value: a positive integer.

11.7.2 Environment

ambient_noise: the ambient noise level in decibels for each of BATMAN & ROBIN's warfare theaters.

value: a positive integer.

lambda: one of the values that determines random number distributions for the Lambda-Sigma Jump process (Buoni, 1989).

value: a positive integer.

sigma: one of the values that determines random number distributions for the Lambda-Sigma Jump process (Buoni, 1989).

value: a positive integer.

transmission_loss: a list of transmission loss values in decibels for each nautical mile of range. The first value is the transmission loss at zero nautical miles. The last value is the transmission loss at 119 nautical miles. The formula,

$$TL = 20 \log r + a \times .001,$$

where r is the range in yards (2025 yards per nautical mile), and a is the absorption coefficient, was used to generate the current transmission loss values. See Urick (1983) for a complete description of this formula.

Convergence zones are simulated by lowering the transmission loss values for those ranges where convergence zones occur. The following is a sample list of transmission loss values where the convergence zone values are shown in boldface:

/north_atlantic/transmission_loss "0 69 78 85 90 95 100 104 108 113 117 120 124 128 132 135 139 142 146 149 153 156 160 163 167 170 173 177 180 183 187 190 193 197 200 **69 69 69** 213 216 220 223 226 229 233 236 239 242 246 249 252 255 258 262 265 268 271 274 278 281 284 287 290 293 297 300 303 306 309 312 **101 101 101 101 101** 331 335 338 341 344 347 350 353 357 360 363 366 369 372 375 379 382 385 388 391 394 397 401 404 407 410 413 416 419 422 **120 120 120 120 120 120 120** 447 450 454 457 460 463 466 469 472"

As shown in the above example, there must be a list of transmission loss values for each of BATMAN & ROBIN's warfare theaters.

value: a list of positive integers.

11.7.3 Sonobuoys

depth: the specifiable depths in feet below sea level that a sonobuoy type can descend.

value: either a list of discrete numeric values or a range of values. For example, the SSQ-53 can be laid at one of three discrete depths, as in:

/SSQ-53/depth "90 400 1000",

while the SSQ-36 can be directed to a range of values, as in:

/SSQ-36/depth "0..1000"

dipping: whether or not a sonar can be dipped and retracted from helicopters.

value: y if the sonar can be dipped; n otherwise.

life: specifiable minutes indicating a sonobuoy type's longevity.

value: a list of numeric values or a range of values.

mode: indicates whether a sonobuoy is active or passive.

value: active or passive.

power: the power in watts used to energize a sonobuoy.

value: a real number.

rf_channels: available radio channels on which a sonobuoy type can communicate.

value: a list of numeric values or a range of values.

11.8 Icon Parameters

The following parameters identify icons used by BATMAN & ROBIN. The ***value*** of each is the name of the file that contains the icon.

air_explo_pic: the icon that depicts the destruction of air platforms in BATMAN.

air_pic: the NTDS (Navy Tactical Data System) icon for Blue and Red air appearing in the right strip in BATMAN used to view, or not view, the air battle.

air_radar_pic: the Blue and Red NTDS air icon appearing in the right strip in BATMAN used to turn air radar coverage on or off.

air_status_pic: the NTDS icon for Blue air appearing in the right strip in BATMAN used to view the status of aircraft for TFA, TFB, or TFC.

alert_pic: the icon used to access the placement of aircraft on Alert or Ready 5, 15, and 30.

base_pic: The airfield-base icon used by Blue, Red, and Green forces

base_rotate_pic: the icon used to rotate between the available *home-base* platforms in ROBIN.

batman_intro_pic: the header or preceding display shown before BATMAN.

big_us_flag: the American-flag icon used in the select "Performance Measures" interface.

big_us_ussr_flag: the American/Soviet-flags icon used in the select "Performance Measures" interface.

big_ussr_flag: the Soviet-flag icon used in the select "Performance Measures" interface.

cap_pic: the CAP (Combat Air Patrol)-station icon appearing in the right strip during the deployment phase of BATMAN.

chaff_pic: the icon used to depict chaff in BATMAN & ROBIN.

chain_pic: the chainsaw icon appearing in the right strip during the deployment phase of BATMAN.

checks_pic: the icon used to depict the track highlighter during track laying in ROBIN.

clear_pic: the icon that depicts the screen-clear operation during deployment in BATMAN.

emcon: The vertical-EMCON icon, i.e., "EMCON" written vertically, used to turn on the radars of a task-force during BATMAN.

esm_status: The icon used to display the ESM status window during BATMAN.

grey25_pic: an icon that provides highlighting in ROBIN. When the user views warfare theaters by selecting areas from the miniature world map in the upper-left corner of ROBIN, the **grey25_pic** is used to highlight the user's current selection.

grey75_pic: an icon used as a pattern for drawing the arrows attached to the zoom box.

grid_pic: the icon used to access the specification of the vector-logic grid radius.

hammer_pic: the Soviet hammer-and-sickle icon used to bring in the Red force in BATMAN.

hit_explo_pic: the icon that depicts a weapons hit against a platform.

hook_pic: an icon of the small box used to enclose the number of Red aircraft in a swarm at a particular point in ROBIN.

horizontal_line: an icon of a horizontal line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

horizontal_shadow: an icon of a horizontal drop-shadow line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

lat_lon_pic: the icon used to go directly from loadout to deployment, bypassing grid specification.

loadout_map_pic: the icon used to return to the warfare theater from loadout in BATMAN.

loadout_next_pair_pic: the icon used to display the next pair of aircraft on a carrier or other *home base* to be loaded out in BATMAN.

move_pic: the icon that represents moving or changing positions of platforms, CAPs, and Chainsaws during deployment.

nprdc_logo_pic: the NPRDC logo icon.

ntds_blue_air_symbol: the NTDS icon for Blue- or Console-force aircraft.

ntds_blue_sub_symbol: the NTDS icon for Blue submarines.

ntds_blue_surf_symbol: the NTDS icon for Blue surface platforms.

ntds_red_air_symbol: the NTDS icon for Red- or path-force aircraft.

ntds_red_sub_symbol: the NTDS icon for Red submarines.

ntds_red_surf_symbol: the NTDS icon for Red surface platforms.

past_esm: The icon used to display lost ESM contacts during BATMAN.

port_pic: The port icon used by Red and Green forces.

radar_toggle: The EMCON-icon displayed that, when engaged, allows the user to turn on individual platform's radar during BATMAN.

remove_pic: the icon that represents erasing platforms, CAPs, and Chainsaws during deployment.

robin_intro_pic: the introductory title to ROBIN.

search_radar: The icon used to depict Red and Green force radar during path specification in ROBIN.

single_status_pic: the icon used to turn on single platform status or fixment during BATMAN. When single-status mode is on, selecting a platform on the map will display its status window. When single-status mode is off, selecting a platform on the map will go on board the platform and display its launch panel.

sm_blue_air_ntds: a small Blue-force air NTDS icon.

sm_blue_sub_ntds: a small Blue-force subsurface NTDS icon.

sm_blue_surf_ntds: a small Blue-force surface NTDS icon.

sm_green_air_ntds: a small Green-force air NTDS icon.

sm_green_sub_ntds: a small Green-force subsurface NTDS icon.

sm_green_surf_ntds: a small neutral- or Green-force surface NTDS icon.

sm_red_air_ntds: a small Red-force air NTDS icon.

sm_red_sub_ntds: a small Red-force subsurface NTDS icon.

sm_red_surf_ntds: a small Red-force surface NTDS icon.

sm_world_pic: the miniature world map used to select warfare theaters in ROBIN.

small_cap_pic: the icon displayed where the user positions CAP-stations during BATMAN.

small_chain_dot_pic: an icon of a small dot representing the end-points of a chainsaw during BATMAN.

sonar_pic: the icon appearing in the right strip in BATMAN containing Blue air, surface, and subsurface and Red subsurface NTDS symbols used to display sonar coverage.

status_pic: the icon used in BATMAN to display the status of a task force's air or surface platforms.

subsurf_explo_pic: the icon that depicts the destruction of subsurface platforms in BATMAN.

subsurf_pic: the Blue and Red NTDS subsurface icons appearing in the right strip in BATMAN used to view, or not view, the subsurface battle.

surf_explo_pic: the icon that depicts the destruction of surface platforms in BATMAN.

surf_pic: the Blue and Red NTDS surface icons appearing in the right strip in BATMAN used to view, or not view, the surface battle.

surf_radar_pic: the Blue and Red NTDS surface icons appearing in the right strip in BATMAN used to turn surface radar coverage on or off.

surf_status_pic: the NTDS icon for Blue surface platforms appearing in the right strip in BATMAN used to view the status of surface platforms for TFA, TFB, or TFC.

task_force_pic: a generic task force icon appearing in BATMAN during aircraft loadout used to view again ships in a task force.

tfa_status_pic, **tfb_status_pic**, and **tfc_status_pic**: the icons appearing in the right strip in BATMAN used to specify which task force's status to display.

vertical_line: an icon of a vertical line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

vertical_shadow: an icon of a vertical drop-shadow line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

view_radar: The icon used to depict Red and Green force radar during the ROBIN viewer.

vl_pic: the Victor-Lima (V/L) icon used to indicate the defended point.

warnings_pic: The warnings icon used to change Blue-force warnings and weapons status during

BATMAN.

weapons_radar: The icon used to depict TAR.

zoom_pic: the icon in BATMAN that represents the zoom function.

11.9 System-Configuration Parameters

big_font: the large font used in BATMAN & ROBIN, e.g., in task force status windows or boards.

value: the name of the file that contains the font.

colormap: the color map used in BATMAN & ROBIN which is viewed as a palette of colors each represented by three numbers that specify the red, green, and blue hues.

value: an absolute pathname to the file containing the color map.

data_path: the complete pathname to the directory that contains BATMAN & ROBIN scenarios.

value: an absolute pathname to a directory.

debug_on: Whether or not debugging error messages should be printed during BATMAN & ROBIN. Generally set to y by the programmer during debugging.

value: y or n.

demo_scenario_number: the number of the scenario that is presented in the BATMAN & ROBIN demonstration.

value: the identification number of an existing scenario.

display_height: the vertical resolution of the display window in pixels.

value: a number between and including 0 and 1152. The recommended value is 1142.

display_left: the x location of the upper-left corner of the display window for BATMAN & ROBIN specified as a pixel offset from (0,0).

value: a number between and including 0 and 1152. The recommended value is 0.

display_top: the y location of the upper-left corner of the display window for BATMAN & ROBIN specified as a pixel offset from (0,0).

value: a number between and including 0 and 900. The recommended value is 0.

display_width: the horizontal resolution of the display window in pixels.

value: a number between and including 0 and 900. The recommended value is 870.

escort_dist_back_tenth_nm: The distance from behind that an escort platform should be from the platform it's escorting.

value: a positive integer (nautical miles).

escort_dist_side_tenth_nm: The distance to the side that an escort platform should be from the platform it's escorting.

value: a positive integer (nautical miles).

escort_threat_dy: The number of **escort_threat_units** in the y-direction that an escort platform should be from the platform it's escorting.

value: a positive integer.

esm_lost_contact_remove_dist: the distance in nautical miles that a lost ESM contact should travel at its current velocity before it is removed from the BATMAN display.

value: a positive integer.

esm_show_blue_tar: Whether or not Blue-force TAR should be displayed during BATMAN.

value: y or n.

esm_status_remove_dist: the distance in nautical miles that a lost ESM contact should travel at its current velocity before it is removed from the ESM status window.

value: a positive integer.

fighter_sector_radius: The default sector radius for platforms flying a BLUE_FIGHTER_AIRCRAFT_MISSION.

value: To date, 250 nautical miles.

fonts: the complete pathname to the directory that contains the fonts used in BATMAN & ROBIN

value: an absolute pathname to a directory.

green/air: the NATO names of the Green aircraft.

value: a list of Green air NATO names separated by spaces.

green/fighter: the NATO names of the Green fighter aircraft.

value: a list of Green fighter NATO names separated by spaces.

green/ship: the NATO names of the Green surface platforms.

value: a list of Green surface NATO names separated by spaces.

green/sub: the NATO names of the Green subsurface platforms.

value: a list of Green subsurface NATO names separated by spaces.

grid_max_radius: the maximum radius of the vector-logic grid.

value: a positive integer larger than grid_min_radius.

grid_min_radius: the minimum radius of the vector-logic grid.

value: a positive integer smaller than grid_max_radius.

guide_names: a list of the home-base platforms available in BATMAN & ROBIN.

value: a list of platforms separated by spaces, e.g.:

/sysparm/guide_names "Nimitz Tarawa Base"

heap_debug_level: a debugging tool used to flag dynamic memory allocation errors. It is intended to be used by the software-maintenance engineer, and requires an understanding of the C *malloc_debug* function (SunOS Reference Manual, 1990).

value: set this value to 0 for *Level 0 malloc_debug* error checking, 1 for *Level 1*, or 2 for *Level 2*.

host_passwd: a security measure used to insure that BATMAN & ROBIN only run on designated host computers.

value: a string of characters.

in_color: whether BATMAN & ROBIN is to be run in color or black-and-white.

value: y for color; n for black-and-white.

itroff_command: the Unix command used to print performance measures on a laser printer.

value: a valid Unix print command, e.g., "ptroff" (assuming *enscript* software loaded).

map_font: the font used to write latitude and longitude labels, e.g., 40.00n.

value: the name of the file that contains the font.

master_passwd_file: the file that contains the system-administrator's password.

value: an absolute pathname to the file.

med_font: the medium-size font used in BATMAN & ROBIN.

value: the name of the file that contains the font.

mother_names: a list of the *mother* or *home-base* platforms in BATMAN & ROBIN.

value: a list of platforms separated by spaces, e.g.:

/sysparm/mother_names "Nimitz Tarawa Base"

msec_engine_update_interval: the amount of processor time in milliseconds dedicated to each call of *simulation_engine* before it returns control to the SunView windowing and input system (see Section 8.4.3, **Engine**).

value: the recommended value is 15.

num_parts: the number of horizontal partitions BATMAN's display is divided into for refreshing purposes in order to enhance the system's response time.

value: a positive number. The recommended value is 10.

pattern_preview_usecs: the number of microseconds that a sonobuoy-pattern preview is displayed.

value: a positive integer. The recommended value is 500000, i.e., one half of a second.

pic_path: the complete pathname to the directory that contains BATMAN & ROBIN icons and graphic objects.

value: an absolute pathname to a directory.

players_directory: the complete pathname to the directory that contains all user files.

value: an absolute pathname to a directory.

printer: the Unix device name of the printer where performance measures are sent for hard copy.

value: a valid Unix device name for a printer, e.g., lw.

red/bomber: the NATO names of the Red bomber aircraft.

value: a list of Red bomber NATO names separated by spaces.

red/fighter: the NATO names of the Red fighter aircraft.

value: a list of Red fighter NATO names separated by spaces.

red/ship: the NATO names of the Red surface platforms.

value: a list of Red surface NATO names separated by spaces.

red/sub: the NATO names of the Red subsurface platforms.

value: a list of Red subsurface NATO names separated by spaces.

scale_font: the font used for the map's scale.

value: the name of the file that contains the font.

sm_font: the small font used in BATMAN & ROBIN.

value: the name of the file that contains the font.

stats_title_font: the font used for the "Performance Measures" title.

value: the name of the file that contains the font.

time_to_stop: the amount of simulated or warped time in minutes that BATMAN will run before stopping.

value: a positive integer.

warn_message_font: the font used for the advisory or warning messages that are displayed prior to BATMAN or ROBIN.

value: the name of the file that contains the font.

11.10 Performance-Measures Parameters

heads_up_dist: the distance in nautical miles a Red-force platform must be less than to be within "heads-up" range of a task force.

value: a positive integer.

stats_save_events: whether or not the consequences of users' tactical actions during BATMAN should be saved to compute performance measures.

value: y if the events should be saved; n if they should not be.

stats_save_results: whether or not the performance measures themselves should be saved in the users' directories.

value: y if performance measures should be saved; n if they should not be.

11.11 User-Database Parameters

button_color: the color of panel-buttons to add/delete users.

value: the form string for this parameter should be "color offset". Color is specified in the "color map" file mentioned above; offset indicates a variation in color, e.g., "red 8" would set button_color to the eighth variation of red.

player_panel_border_pic: an icon of a thin dark horizontal line used as a border between the "operations" and "list of users" panels displayed when the user is adding, deleting, or selecting users.

value: the name of the file that contains the icon.

uinfo_border_color: the background color of the screen when entering a new user's name and social security number.

value: a string of the form: "color offset". See **button_color** above.

user_bg_color: the color of empty slots in the "list of users" panel.

value: a string of the form: "color offset". See **button_color** above.

user_fg_color: the color of users' names in the "list of users" panel.

value: a string of the form: "color offset". See **button_color** above.

username_border_color: the border color around users' names in the "list of users" panel.

value: a string of the form: "color offset". See **button_color** above.

11.12 GFED Parameters

The database files **blue_params.db**, **red_params.db**, and **green_params.db** define the characteristics of the Graphical Frontend to the Database (GFED) for Blue, Red, and Green platforms, respectively. In particular, these databases define the changeable parameters and the GFED sub-windows for each platform. As an example, consider the following segment from **blue_params.db**:

/F-14/basic_params "hit_tolerance_level kill_priority radar_cross_section"
/F-14/basic_win_pos "bottom"
/F-14/num_regions "4"
/F-14/params_region1 "speed_max_conserve speed_full_power"
/F-14/x_region1 "0"
/F-14/y_region1 "40"
/F-14/width_region1 "25"
/F-14/height_region1 "20"
/F-14/win_pos_region1 "left"
etc.

This tells the GFED software that there are four "hot-spot" regions for the F-14 (*num_regions*) and gives the characteristics for region1 (*x_region1*, *y_region1*, *width_region1*, *height_region1*, and *win_pos_region1*). Additionally, the default or "basic" region is to be associated with the bottom sub-window (*basic_win_pos*).

A region defines a "hot-spot" on a platform's icon. When the user selects a point within a particular region, the parameters associated with that region are displayed in the corresponding GFED sub-window, e.g., aircraft nose sensors, radar parameters. Regions are defined in terms of percentages relative to (0,0) (i.e., the upper-left corner) on the subject platform icon. Referring back to the F-14 example above, region 1 covers 0% to 25% going from left to right (*x_region1* and *width_region1*), and 40% to 60% going from top to bottom (*y_region1* and *height_region1*). Hence, if the user selects the middle-left portion of the F-14 icon, region 1 will be triggered and its associated sub-window will be displayed. The new parameters added in support of the GFED are described below.

basic_params: The default changeable parameters for the platform. These are listed when the user selects a point on the platform's icon that isn't within any of the defined regions.

value: A list of platform parameters separated by spaces.

basic_win_pos: The location of the default or "basic" sub-window relative to the platform's icon.

value: top, bottom, left, or right.

num_regions: The number of regions defined for the platform.

value: 1, 2, 3, or 4.

params_region[N]: The changeable parameters that should be displayed when region N is selected, where N is 1, 2, 3, or 4.

value: A list of platform parameters separated by spaces.

x_region[N]: The x coordinate that starts region N, where x is a percentage of the platform's icon relative to (0,0) and N is 1, 2, 3, or 4.

value: Between 0 and 100, inclusive.

y_region[N]: The y coordinate that starts region N, where y is a percentage of the platform's icon relative to (0,0) and N is 1, 2, 3, or 4.

value: Between 0 and 100, inclusive.

width_region[N]: The width of region N as a percentage of the platform's icon, where N is 1, 2, 3, or 4.

value: Between 0 and 100, inclusive.

height_region[N]: The height of region N as a percentage of the platform's icon, where N is 1, 2, 3, or 4.

value: Between 0 and 100, inclusive.

win_pos_region[N]: The location of region N's sub-window relative to the platform's icon, where N is 1, 2, 3, or 4.

value: top, bottom, left, or right.

12.0 Scenario Database

The scenario database is comprised of standard text files which are created by ROBIN and used by BATMAN. The Unix utilities "cat" and "more" can be used to view these files. This database is stored in the directory specified by the **data_path** parameter from the Parameter Database.

Each scenario is split into three files: a Blue-force file, a Blue-force messages file, and a Path-force file. These contain all the necessary data for the Blue-, Red-, and Green-forces for a specific scenario. The format of the filenames is "blue", "messages", and "path" followed by a scenario number, e.g., the files for scenario 10 would be **blue.10**, **messages.10**, and **path.10**. Scenario numbers range from 1-999.

In addition to these files, there are five support files used for scenario construction and class-test assignments: **blue.1000**, **blue.2000**, **assignment_index**, **class_index**, and **test_index**. These files are also located in the directory specified by **data_path**. The **blue.1000** and **blue.2000** files are used by ROBIN as a template for building **blue.n** files. The remaining three support files are used to coordinate class-test assignments: the **assignment_index** file identifies tests that have been

assigned to classes; the **class_index** file identifies the students that are in each class; and the **test_index** file identifies the scenarios that are in each test.

12.1 Blue-Force File

The Blue-force file lists air, surface, and subsurface platforms as well as weapons and sonobuoys that are available for allocation, deployment, and management in BATMAN which reads this file to initialize scenario data structures.

The first field in this file is the task-force designator. BATMAN & ROBIN can handle a maximum of three Blue task forces: TFA, TFB, and TFC. Following the task-force designator line, is a list of the platforms in a specific task force. In each line, the number of platforms is listed followed by their NATO names.

If a platform is a *guide* -- Nimitz, Base, or Tarawa -- then it will have a latitude-longitude designation after it. If the latitude-longitude designation is "0.0 0.0", then the guide has not been preset in ROBIN, and the user is free to place it where they choose during deployment. If a platform is a *mother*, i.e., capable of carrying *sub* platforms, then it can have additional resources "on board", which are enclosed between brackets ({ and }).

When the user changes Blue-force parameters for a scenario using the GFED (see Sections 8.10, **Database and Graphical Frontend Packages**, and 11.12, **GFED Parameters**), that change is saved in the corresponding **blue.n** file. The change is tagged with the keyword **PARAMETERS** which is followed by a list of changed parameters and their values. For example, the following segment from a **blue.n** file indicates that the Ticonderoga's **hit_tolerance_level** should be 200:

```
1 Ticonderoga PARAMETERS { hit_tolerance_level 200 }
{
                    2 SH-60B
                    20 mk46
                    200 SSQ-53
                    200 SSQ-62
                    200 SSQ-77
}
```

12.2 Blue-Force Messages File

The messages file contains all of the scenario's messages, including the time they occur and the type for each message.

12.3 Path-Force File

The Path-force file contains a summary of the tactical situation (including a list of the scenario's Red and Green platforms) and track-movement or path definitions for each platform.

12.3.1 Tactical-Situation Section

In a manner similar to the Parameter Database, a tactical situation is described by a set of parameters, each holding a particular value. This provides a uniform method for describing a variety of battle scenarios. The following is a brief description of each parameter from the Tactical-Situation section of a Path-force file.

blue_w_and_w: the scenario's initial Blue-force warnings and weapons status against hostile air, surface, and subsurface platforms.

> ***value***: the warnings and weapons status against air, surface, and subsurface platforms, respectively.

bp: indicates a Red- or Green-force base or port.

> ***value***: the base or port's type, identification number, latitude, longitude, and a list of the platforms that initiate from the base or port.

green_platforms indicates the name and amount of each Green platform defined for the scenario.

> ***value***: a list of Green platforms separated by spaces, where each platform is followed by a positive integer, which indicates how many of a specific type.

red_platforms: indicates the name and amount of each Red platform defined for the scenario.

> ***value***: a list of Red platforms separated by spaces, where each platform is followed by a positive integer, which indicates how many of a specific type.

red_wpn_status: the Red-force's weapons status against Blue-force platforms.

> ***value***: a list of "time, weapons-status" pairs. For example, "0 tight 15 free" would indicate "weapons tight" until the 15th second, at which time the Red-force changes to "weapons free".

time_warp: specifies a time-warp for the scenario.

> ***value***: a positive integer.

vl_latitude and **vl_longitude**: the latitude and longitude of V/L.

> ***value***: real numbers, e.g., (vl_latitude = 15.00, vl_longitude = -80.00). The latitude and longitude of V/L will lie somewhere in the scenario's warfare theater.

warfare_theater: the warfare area for the battle.

> ***value***: at this time the following theaters are available, but not limited to: arabian_sea, bering_sea, caribbean, japan_sea, kamchatka_peninsula, mediterranean, murmansk, north_atlantic, persian_gulf, and south_east_asia.

12.3.2 Path Section

This section contains a series of text lines defining the movement-tracks or paths for Red- and

Green-force platforms in a scenario. Unlike the tactical situation-display parameters, the track definitions are listed on contiguous lines without breaks or comments. Hence, it is difficult for the user to decipher the contents of these records. This track data is designed to be read by BATMAN when the scenario is initialized, or by ROBIN when the scenario is modified.

Platform-track data is represented by a tree structure with nodes and vectors. A node is a location where one or more platforms of the same type move together. Each node may have one or more vectors depicting segments of the movement path for an individual platform or swarms of them. Nodes must also have an origination base or port (*org_bp*), and can have an optional destination base or port (*dst_bp*). The last vector for a node is either the terminal point, or the position where the swarm splits into subswarms or individual platforms. For more information, see Figure 5 in Section 7.0, **BATMAN & ROBIN Global Data Structures.**

When the user changes Path-force parameters for a scenario using the GFED (see Sections 8.10, **Database and Graphical Frontend Packages**, and 11.12, **GFED Parameters**), that change is saved in the corresponding **path.n** file. The change is tagged with the keyword **PARAMETERS** which is followed by a list of changed parameters and their values. For an example, refer to the following Section, 12.3.3, **Sample Path-Force File.**

12.3.3 Sample Path-Force File

The following is an example of a Path-force file.

```
warfare_theater: kamchatka
vl_latitude: 46.580618
vl_longitude: 154.963363
time_warp: 60
blue_w_and_w: air red free surf red free sub red free
red_wpn_status: 0 free
red_platforms: RedSubPort 1 Charlie 1
green_platforms:
bp: plat RedSubPort bp_id 5 lat 46.003428 lon 149.949852 Charlie 1


path_node: plat Charlie start_id 45 end_id 46 start_time 0.000000 org_bp 5
                PARAMETERS { hit_tolerance_level 150 }
path_vector: lat 45.478892 lon 149.846345 altitude -100 speed 15
path_vector: lat 44.672755 lon 150.592310 altitude -100 speed 15
path_vector: lat 45.103233 lon 149.497859 altitude -100 speed 15
```

13.0 Graphic Database

This database contains all the icons or graphic objects used in BATMAN & ROBIN. These are in Sun raster-file format, and each exists in its own file (Pixrect Reference Manual, 1990). This database is stored in the directory specified by the **pic_path** parameter of the Parameter Database.

14.0 User Database

The User-Performance database contains multivariate performance measures and recorded scenarios for the user. Each BATMAN & ROBIN user is given their own Unix directory which contains the user's name and social security number, all performance measures to date, and any scenarios that the user has recorded for later playback. BATMAN & ROBIN can maintain more than names and social security numbers of users, e.g., number of flight hours or other descriptive data.

References

Buoni, F. B. (1989). *The Design and Implementation of Detection Models for the Battle Management Assessment System.* Unpublished masters thesis. Monterey, CA: Naval Postgraduate School.

Butler, C.W., Hodil, E.D., & Richardson, G.L. (1988). Building knowledge-based systems with procedural languages. *IEEE Expert, 3,* 47-59.

Federico, P-A., Bickel, S.H., Ullrich, R.R., Bridges, T.E. & Van De Wetering, B. (1989). *BATMAN & ROBIN: Rationale, Software Design, and Database Descriptions (TN 89-18).* San Diego, CA: Navy Personnel Research and Development Center.

Holtzman, S. (1989). *Intelligent decision systems.* Reading MA: Addison-Wesley.

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1986). Direct-manipulation interfaces. In D. A. Norman & S. W. Draper (Eds.), *User centered system design.* Hillsdale NJ: Lawrence Erlbaum Associates.

Kernighan, B.W. & Ritchie, D.M. (1988). *The C Programming Language, Second Edition.* Englewood Cliffs NJ: Prentice-Hall.

Pixrect Reference manual. (1990). Mountain View CA: Sun Microsystems.

Raeth, P. G. (1990). Process states as decision criteria. *AI Expert, 5,* 40-45.

Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and information technology,* 1, 237-256.

SunOS Reference Manual. (1990). Mountain View CA: Sun Microsystems.

SunView Programmer's Guide. (1990). Mountain View CA: Sun Microsystems.

SunView System Programmer's Guide. (1990). Mountain View CA: Sun Microsystems.

Urick, R. J. (1983). *Principles of Underwater Sound.* New York: McGraw-Hill.

APPENDIX A

AN O(D+(N $\log_2$ N)) ALGORITHM

FOR RANGE/BEARING RESTRICTED SEARCH

IN TWO DIMENSIONS

AN O(D+(N $\log_2$ N)) ALGORITHM FOR RANGE/BEARING RESTRICTED SEARCH IN TWO DIMENSIONS

William Root
Systems Engineering Associates, Inc.

SETTING

Let N "platforms," $P_1,P_2,...P_N$ be arranged in 2-space, such that platform P_k is located at coordinates $<x_k,y_k>$ and aims its sensor in direction a_k. The sensor "detects" any other platform whose distance from P_i is less than or equal to r_k and whose bearing from P_k lies in the range $a_k \pm s_k$.

PROBLEM

Given $<x_k,y_k>$, a_k, r_k, and s_k for each of the N platforms $\{ P_k \mid k = 1,2,..N \}$, construct the list of all pairs j,k such that P_j detects P_k.

SOLUTION

The algorithm is a variant of the "layered range tree" method of Willard and Lueker [WL].

First, construct a list of triples $\{ < k , x_k , y_k > \mid k = 1,2,...N \}$ sorted in order of increasing x-coordinate (primary key) and increasing y-coordinate (secondary key). In an actual implementation, the algorithm may select from among insertion sort, shellsort, and heapsort to perform this sort, depending on the cardinality N of the platform set. Use of heapsort in all cases guarantees that this phase of the algorithm is O(N $\log_2$ N) . (Note: In applications of this algorithm to tactical air gaming, empirical data suggests that the number of new inversions among aircraft x-coordinates per unit of time as a function the number N of aircraft has order $\leq$ O(N) ; a sort such as the Cook-Kim sort [CK] may be used to advantage in this phase of the algorithm, given such an assumption) .

Next, construct a *heap structure* of 2N-1 nodes in which the key values are *pairs* $x_{lower} : x_{upper}$ of x-coordinate values from the above list. This heap structure -- referred to as the "primary heap structure" -- is easily implemented as an array of N contiguous records using the customary embedding. Assign the x- coordinates in the original list, left to right, in increasing order, as the key values in the *leaf nodes* of the primary heap structure. That is, each leaf node should contain a pair of the form $x_j : x_j$ where $< x_j , y_j >$ is the location of platform P_j for some $1 \leq j \leq N$. This phase of the algorithm is clearly O(N).

The *internal* nodes of the primary heap structure are assigned key values as follows: x_{lower} in the parent node equals x_{lower} in the *left* child node; x_{upper} in the parent node equals x_{upper} in the *right* child node. Each node in the primary heap structure thus corresponds to an *interval* on the x-axis. This phase of the algorithm is clearly O(N-1), and produces a completed structure which is both a heap structure and a binary search tree with respect to x-coordinates of platforms.

Next, the primary heap structure is traversed bottom-up, right-to-left, and for each internal node P with key value pair $x_{lower} : x_{upper}$, a "y-list" of pairs $\{ < k , y_k > \mid x_{lower} \leq x_k \leq x_{upper} \}$ is constructed and sorted in increasing order of y_k by merging the sorted y-lists of the left and right children of P. Initially, the y-list associated with each leaf node with key value pair $x_j : x_j$ is simply the singleton $\{ < j , y_j > \}$. Simultaneously, we associate with each element y_p of the parent y-list two "y-pointers" into the left and right children's y-lists; namely, to that unique minimal element $y_{p'}$ in the left child y-list for which $y_p \leq y_{p'}$, and to that unique minimal element $y_{p''}$ in the right son y-list for which $y_p \leq y_{p''}$. It is clear that the y-list associated with the root of the primary heap structure is itself the linear list equivalent of a heap structure with the binary search property. This list is referred to as the "y-heap". Since the nodes at any fixed depth in the primary heap structure correspond to a partition of some subset (possibly improper) of the set of platforms, each y-coordinate appears in at most one y-list at each depth. Therefore the total number of y-coordinates in all y-lists at any fixed depth is N. Since the depth of the primary heap structure is $O(\log_2 N)$, this phase of the algorithm is at worst $O(N \log_2 N)$.

Now suppose we must determine all detections by platform P_k.

We first determine all platforms which lie in the *square* of side $2r_k$ centered at $< x_k , y_k >$. We start by searching the y-heap for the minimal y_M such that $y_k - r_k \leq y_M$. Next we search the primary heap structure for that node with key value pair $x_{lower} : x_{upper}$ for which x_{lower} is minimal such that $x_k - r_k \leq x_{upper}$. As we descend through the primary heap structure searching for this node, we simultaneously descend with pointer q through the tree of y-pointers rooted at y_M, mimicking exactly the sequence of left and right branches we perform in the primary heap structure. Each time we branch *left* in the primary heap structure from node w, we process as "detected" all platforms whose y-coordinates appear at or after the target of the right link from q into the y-list of the right-son of w (and thus, which lie within the required circumsquare of the radar detection radius of P_k; whether the platform in question lies within the pie-shaped radar detection sector of P_k itself can be immediately determined by a vector-algebraic algorithm of O(1)). Since at most one y-list is processed at each depth in the primary heap structure, this subphase of the algorithm is at most $O(D_{1k}+(\log_2 N))$ where D_{1k} is the total number of detections during the descent through the primary heap structure.

Finally, we repeat the above process, but this time searching the primary heap structure for that node with key value pair $x_{lower} : x_{upper}$ for which x_{lower} is maximal such that $x_k - r_k \leq x_{upper}$. The order of this subphase of the algorithmn is at most $O(D_{2k}+(\log_2 N))$ where D_{2k} denotes the number of detections during this descent through the primary heap structure.

Combined, the order of the order of the above two subphases of the algorithm is at worst $O(D_k+(\log_2 N))$, where $D_k = D_{1k} + D_{2k}$ denotes the total number of platforms detected by platform P_k. Referring now to the original problem, it is clear that repeating the above process once for each of N platforms will yield at worst an $O(D+(N \log_2 N))$ algorithm, where D is the total number of detections by all platforms.

REFERENCES

[CK] COOK,C.R., and D.J.KIM: "Optimal Sorting Algorithms for Nearly-Sorted Lists", *Comm. ACM*, 23 (11), Nov. 1980.

[WL] LUEKER,G.S.: "A Data Structure for Orthogonal Range Queries", *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pp. 28-34, 1978.

APPENDIX B

BATTLE-DAMAGE-ASSESSMENT SIMULATION

BATTLE-DAMAGE-ASSESSMENT SIMULATION

1.0 Description

BATMAN & ROBIN simulate battle-damage assessment by assigning numerical values to platforms and weapons. The numerical value assigned to a platform is called its **hit_tolerance_level**, and the numerical value assigned to a weapon is called its **damage_pts**. These values are used to estimate battle damage, and are stored in BATMAN & ROBIN's Parameter Database. The numerical values used for **hit_tolerance_level** and **damage_pts** were obtained from the *Harpoon* board game (Bond, 1987), or intelligently estimated based upon similar platforms or weapons.

When a weapon hits a platform, it inflicts specified **damage_pts** against the platform. The **damage_pts** are added to the platform's current battle damage. In order to not degrade BATMAN & ROBIN's performance, the computations required to simulate battle-damage assessment are based upon a simplistic, but efficient, mathematical model obtained from the Naval Postgraduate School. It is stated below.

$$\text{percent-of-damage} = [(\text{current-damage} / \textbf{hit\_tolerance\_level})^2 * 100]$$

As can be seen, a platform's damage increases exponentially as its current damage rises. This simplistic model could easily be enhanced to provide more sophisticated battle-damage assessment, including platform-speed degrading, submarine surfacing, and gradual platform sinking. Other details like flooding or fire damage could also be modeled.

2.0 Assumptions and Constraints

The following assumptions and constraints were made during the design of the battle-damage-assessment simulation:

- A platform is destroyed when its current damage reaches or exceeds its **hit_tolerance_level**, i.e., when its damage percentage becomes greater than or equal to one-hundred percent. To date, damage percentages are displayed in BATMAN's task-force and individual-platform status windows.
- The **hit_tolerance_level** for all air platforms is one. That is, air platforms will be destroyed the first time they are hit.
- The Blue-force's **mk48** and the Red-force's **torptypec1** inflict a higher **damage_pts** value against subsurface platforms than they do against surface platforms.

Reference

Bond, L. (1987). Harpoon: Modern Naval Wargame Rules -- Data Annex Book. Bloomington, IL: Game Designers' Workshop.

APPENDIX C

RELATIONAL DATABASES: CONSIDERATIONS, ISSUES, AND EVALUATION

RELATIONAL DATABASES:

CONSIDERATIONS, ISSUES,

AND EVALUATION

Randy R. Ullrich and Thomas E. Bridges
Systems Engineering Associates, Inc.

1.0 Purpose and Scope

This report assesses BATMAN & ROBIN's current and future database needs, compares them against candidate Database Management Systems (DBMS's), and makes selection recommendations.

BATMAN & ROBIN store their data in four separate databases:

1. Object-Definition (i.e., parameters for platforms, weapons, sensors, etc.).
2. Scenario.
3. Graphic.
4. User.

Although this study focuses on DBMS's for improving interactions with the Object-Definition Database, appropriate DBMS's could facilitate BATMAN & ROBIN in coordinating all four of its databases.

The Object-Definition Database is currently implemented using Sun's User Defaults Database tool (SunView System Programmer's Guide, 1988). It was originally decided to use this tool because it was free, relatively efficient, and easy for programmers to use. However, as the functionality and complexity of BATMAN & ROBIN have increased, so have the demands on the Object-Definition Database This report formally explores DBMS alternatives that may be more appropriate for BATMAN & ROBIN.

The reader is assumed to have a basic understanding of the design of BATMAN & ROBIN (Federico, Bickel, Ullrich, Bridges, and Van De Wetering, 1989), as well as familiarity with DBMS concepts (Ulrika, 1990), SunOS (SunOS Reference Manual, 1988), and the C programming language (Kernighan and Ritchie, 1988).

The next section identifies limitations with the Object-Definition Database and the issues affecting this report's recommendations. Section 3 lists the candidate DBMS's and section 4 describes the criteria used to evaluate them. Section 5 evaluates the DBMS's. Finally, Section 6 presents conclusions formed by this study and Section 7 offers database-selection recommendations.

2.0 Introduction

The Object-Definition Database is currently implemented with Sun's User Defaults Database. This database is a standard text file stored in ASCII format. Sun provides an editor (**defaultsedit**) to

modify the database. Application programs access this database through a set of SunOS functions.

There are a number of limitations with this implementation:

- It is awkward for non-technical users to add, delete, and modify objects in the database. BATMAN & ROBIN's Object-Definition Database has grown past the limits of Sun's **defaultsedit** editor, so a text editor (e.g., **vi**) must be used to modify it. Most non-technical users are unfamiliar with UNIX editors and must learn one before they can modify the database.

- It is misusing Sun's User Defaults Database tool, which is intended as a centralized location to maintain customization information for UNIX programs (e.g., **mail**, **csh**, **sunview**, etc.). For example, the User Defaults Database entry:

  /Mail/Set/Askcc "Yes"

  tells the **mail** program to automatically prompt for the "cc" field when sending a message. BATMAN & ROBIN, however, obstruct developers from using the User Defaults Database for this purpose.

- It can contain only text data. The User Defaults Database is stored in ASCII format and cannot store binary data, e.g., icons.

- The programming functions provided for accessing the User Defaults Database are limited to basic search and put routines.

Compounding these limitations are a number of major future enhancements being considered for BATMAN & ROBIN. The most notable are:

- **An Object-Definition Database Editor**: Since the platform, weapon, sensor, and other related parameters in the Object-Definition Database are unclassified and only approximations, users should be able to change them to their more accurate classified values. To ease users' editing task, an Object-Definition Database editor with a user-friendly interface, fashioned after BATMAN & ROBIN, will be written.

- **Additional Computer Models**: A number of DoD sites will use BATMAN & ROBIN as an interface to their computer models. To do this, they need to add and delete entries from the Object-Definition Database. Therefore, candidate DBMS's should not only facilitate changes to existing attributes in the Object-Definition Database, but should also facilitate unlimited additions and deletions.

- **Multiple-Machine Architecture**: Currently, BATMAN is executed by one user on a Sun workstation. The user allocates, deploys, and manages Blue-force assets against a hostile Red threat. BATMAN has proved effective in presenting and coordinating this resource-allocation problem to its user. As a possible improvement to BATMAN, the addition of players and Sun workstations has been suggested. Under this architecture, multiple machines connected by a network would allow any number of users to be involved in the battle. For example, one user might manage the Blue force at one workstation while a second user at a different workstation managed the Red force.

Therefore, the candidate DBMS's should be able to operate in this environment.

- **Intelligent Platform Behavior**: This proposed enhancement would allow platforms to make their decisions dynamically based on the state of the battle and the nature of the threat. It is addressed in detail as an evaluation criterion. For more information, refer to section 4.4, **Artificial Intelligence.**

3.0 Candidate DBMS's

Table 1 lists the DBMS's evaluated in this report, and identifies the source that brought each system to this report's attention. Addresses and telephone numbers of the DBMS vendors are listed at the end of this document. The information in this report was gathered by direct consultation with vendors, through reading printed material, and by personal experience.

Table 1

Candidate DBMS's

DBMS, Vendor	Source
Informix, Informix Software, Inc.	UNIX Database Management Systems
Ingres, Relational Technology, Inc.	Statement of Work
Oracle, Oracle Corp.	Statement of Work
Sybase, Sybase, Inc.	Statement of Work
G-Base, Graphel	Statement of Work, UNIXWORLD
Gemstone, Servio Logic Corp.	Statement of Work, UNIXWORLD
Ontos (V-Base), Ontologic Corp.	Statement of Work, UNIXWORLD
Vision, Innovative Systems	Statement of Work, UNIXWORLD
C Database Toolkit, Jaybe Software	Catalyst Catalog
C-ISAM, Informix Software, Inc.	Catalyst Catalog
C-tree, Faircom	The Programmer's Shop
db_VISTA III, Raima Corp.	Catalyst Catalog
ndbm, Sun Microsystems, Inc.	SunOS (4.3 BSD UNIX)
User Database, Sun Microsystems, Inc.	Current system

These DBMS's can be grouped into three main categories: relation database management systems (RDBMS's), object-oriented database management systems (OO-DBMS's), and database management toolkits. The following sections describe the capabilities of each.

3.1 Relational Database Management Systems (RDBMS's)

RDBMS's are designed as standalone, multi-user, database environments. They excel in handling large sophisticated databases, e.g., on-line transaction processing (OLTP) systems, and decision

support systems. Some example applications include banking, finance, accounting, and factory automation. The candidate DBMS's in this category are Informix, Ingres, Oracle, and Sybase.

These systems provide capabilities to create and interact with databases, generate reports, and write high-level language application software. The following list describes typical features of RDBMS's.

- **Menus and Forms**: Allow application programmers to build menu- and form-driven front-ends to their databases.
- **Reports**: Allow users to synthesize the data in the database and to print reports based on various criteria.
- **SQL (Structured Query Language)**: A language for extracting information from the database. Originally developed by IBM, this language is now an industry standard. Its English-like syntax makes it easy to use. For example, the command: "select name from address where state = CA" would print all California residents from an address database.
- **4GL (Fourth Generation Language)**: A database programming language that allows programmers to build applications on top of the database, generally with fewer instructions than traditional high-level programming languages.
- **Programming Interface**: A function library or an SQL preprocessor that allows C programs to search the database and extract values.

3.2 Object-Oriented Database Management Systems (OO-DBMS's)

OO-DBMS's, fashioned after object-oriented programming concepts (Meyer, 1988), model an object in the real world with a corresponding item in the database. Both the object's attributes and its behavior can be stored. This is similar to the object-oriented programming scheme of encapsulating a data structure and the functions that manipulate that data structure into one module. Inheritance, where the attributes of one object are used to build another object, is also supported by OO-DBMS's. Another powerful feature of OO-DBMS's is that they can store abstract binary data (e.g., bit-maps, scanned images, video), whereas typical RDBMS's can store only predefined data types (e.g., character strings, integer values, real values, dates). Most of the target markets of OO-DBMS's have been scientific and technical, however, they could be applied to many fields, including financial.

This object-oriented perspective is consistent with BATMAN & ROBIN's software design and could provide additional clarity and elegance to the software. For example, an OO-DBMS could ease the chore of retrieving an F-14 platform from the database. Currently, platform attributes and icons are stored in separate databases, requiring separate access routines. Using an OO-DBMS, however, BATMAN & ROBIN would have to make only one query. In response, the OO-DBMS would return attributes of the F-14 (text data), bit-mapped icons of the F-14 (binary data), and rules governing the F-14's behavior (code). This same scheme could be applied to BATMAN & ROBIN's other objects, such as weapons, sensors, and jamming pods.

A disadvantage of OO-DBMS's is that they are typically unproven, developed by young start-up

companies whose stability is questionable. Additionally, OO-DBMS's tend to be slower than RDBMS's, and they lack the wide range of support utilities provided by RDBMS's.

This report's information on the four OO-DBMS's (G-Base, Gemstone, Ontos, and Vision) came from the UNIXWORLD article "Object-Oriented Databases Arrive" (the Ontologic product discussed in the article, V-Base, has been renamed to Ontos). Addresses and telephone numbers of Innovative Systems (Vision) and Graphel (G-Base) were unavailable, even after consultation with UNIXWORLD. Because these vendors were unreachable, their products are not discussed in this report. However, the other companies mentioned in the article, Servio Logic (Gemstone) and Ontologic (Ontos), were found, and their products are included.

3.3 Database Management Toolkits

Database management toolkits are libraries that provide simple database creation and access functions. They usually provide an interface to C applications. The candidate DBMS's in this category are The C Database Toolkit, C-ISAM, C-tree, db_Vista III, ndbm, and User Defaults Database (i.e., the current system).

All database management toolkits use the Indexed Sequential Access Method (ISAM) technique of data management. The most prevalent ISAM indexing method uses a b-tree (Wirth, 1976) due to its efficiency and its ability to easily traverse databases in sorted order. Some database toolkits also provide add-on SQL, screen management, and reporting modules, providing capabilities similar to RDBMS's.

4.0 Criteria

The following sections define the criteria used to evaluate the candidate DBMS's. These were gathered from the Statement Of Work and from consultation with the software developers. While researching this report, however, it was discovered that some of these criteria have no effect on the database selection. In those cases, the reasons why the selection is not effected are explained.

4.1 Human-Computer Interface

BATMAN & ROBIN's human-computer interface is based on a strict direct-manipulation protocol that makes the system easy to use. This protocol includes the following guidelines:

- All user interactions come from the mouse; the keyboard is used only when absolutely necessary.
- Graphic representations, i.e., icons, are used to depict objects. Using the mouse, the user "directly manipulates" objects on the screen.
- The system presents objects on the screen only when appropriate. That is, the system itself poses no obstacle; the logical actions that the user can make are displayed for him automatically.

Many DBMS's come with an integrated set of application development tools, e.g., screen-form utilities. These utilities help users define and modify their databases. However, none of them adheres to BATMAN & ROBIN's strict human-computer-interface protocol as outlined above. This is precisely why a custom Object-Definition Database editor will be written. Since this editor

can be created using any DBMS with a C-language interface, this criterion is not an issue and is not considered in the evaluation.

4.2 Software Interface

This criterion poses two questions:

1. How much software will have to be written or changed to integrate the DBMS into BATMAN & ROBIN?
2. Can the DBMS store binary data, i.e., could it be applied to BATMAN & ROBIN's Icon database?

For example, question one asks whether a candidate DBMS would require merely rewriting the current database-interface routines *d_get_i* and *d_get_s* or whether more work would have to be done. The estimated number of source-code lines to be written or changed is used as the metric for answering this question.

DBMS's judged against this criterion produce evaluations like "100 lines / YES", indicating that approximately 100 source lines would have to be added or changed to integrate the DBMS into BATMAN & ROBIN and that the DBMS can be applied to BATMAN & ROBIN's Icon database.

4.3 C Compatibility

BATMAN & ROBIN is written in the C programming language and is targeted at the Sun family of computers. All candidate DBMS's must contain a C programming-language interface; support Sun-3, Sun-4, and SPARCstation computers; and operate in the SunView windowing environment under SunOS Version 4.0 or later.

4.4 Artificial Intelligence

Currently, all platform actions either occur automatically (e.g., weapon firing) or require manual intervention (e.g., platform movement). A future objective of BATMAN & ROBIN is to incorporate intelligent-platform behavior. Briefly, this capability allows platforms to make their decisions based on defense conditions (DEFCONS) using rules of engagement (ROEs). A knowledge base, derived from consultation with tactical experts, would be integrated into BATMAN & ROBIN to drive the decision process. At this time, platform behavior has not been established, so the design of this feature cannot be finalized. There are, however, two basic ways to represent the intelligent behavior: as code or as data.

The first design alternative, representing intelligent behavior as code, is straightforward to implement. The rules of behavior regarding movement and weapon firing would be transcribed in C code. This approach is compatible with the current BATMAN & ROBIN design, since each platform has functions for drawing, moving, detecting, and firing weapons. Currently, many different types of platforms use the same functions. For example, F-14s, F/A-18s, and S-3Bs all use the same weapon-firing function. To incorporate intelligent platform behavior, each type of platform would have its own specific routines to model its behavior.

If BATMAN & ROBIN were to implement intelligent behavior as code, it would have little effect on the DBMS recommendation. The only database requirement would be to store the names of the

control functions for each platform, and any DBMS under consideration is capable of this.

The second approach is to represent intelligent behavior as data. There are two options for this approach: (1) write a custom expert system or (2) use a commercially available expert-system shell.

A DBMS would help when writing a custom expert system, since it could store the knowledge base with the platform data. However, writing an expert system would be costly and time consuming. With powerful tools already available, this would not be cost effective for BATMAN & ROBIN. Writing a custom expert system is not considered a reasonable option and therefore plays no part in the DBMS recommendation.

The second approach to implementing intelligent behavior as data is to use an expert-system shell. Most expert-system shells provide an inference engine to evaluate the knowledge base, an interface for creating and maintaining the knowledge base, a debugging tool to test the knowledge base, and tools to create an end-user interface (Gevarter, 1987). All expert-system shells store the knowledge internally; some even provide software interfaces to allow programs to access their information. Since the expert-system shell maintains the data itself, the DBMS recommendation again is not affected.

In summary, intelligent-platform behavior has no effect on this report's recommendations, regardless of whether this behavior is implemented as code or as data. Therefore, this criterion is not included in the evaluation.

4.5 Defaults Conversion

The format of the Object-Definition Database (i.e., the ".defaults file") is specific to Sun's User Defaults Database. While many candidate DBMS's provide conversion routines, none provides automatic conversion from the User Defaults Database format. For all DBMS's, a program must be written to read in the ".defaults file" and to write out the appropriate DBMS records.

This conversion would be a simple programming task. Currently, BATMAN & ROBIN uses the routines *d_get_s* and *d_get_i* to retrieve data from the Object-Definition Database (the *d_get_s* routine gets a string value; the *d_get_i* routine gets an integer value). For example, the call:

d_get_s("F-14", "long_name", plat->long_name)

gets the long name of the F-14 (i.e., Tomcat) and stores it in the *long_name* field of a platform record.

As currently planned, the interface between BATMAN & ROBIN and the Object-Definition Database would not change; *d_get_s* and *d_get_i* would still be used, but would be rewritten to interface with the selected DBMS. A program will convert the Object-Definition Database using the old *d_get_s* and *d_get_i*. The program will read all the fields for platforms, weapons, and detection units and will then write them out in the new DBMS's format. Once the new database has been created, the old *d_get_s* and *d_get_i* in BATMAN & ROBIN would be replaced with their new versions.

In summary, any candidate DBMS with a C programming interface would support this simple

conversion program. Therefore, this criterion is not included in the evaluation.

4.6 Vendor Support

Each vendor should provide, at the minimum, telephone support. That is, the vendor would have a dedicated group to answer questions and provide help. Additionally, vendors who are able to provide on-site support in San Diego would be looked at favorably.

4.7 Product Stability

This report considers a product stable when it (1) has been on the market for at least three years, and (2) has sold at least 250 units.

4.8 Cost

There are two DBMS costs to consider: (1) development-system cost, incurred by Navy Personnel Research and Development Center (NPRDC) and by transition sites, and (2) end-user cost. A transition site uses BATMAN & ROBIN source code as a point of departure for other related Navy training projects, and modifies the code to meet their needs. End-user sites do not receive source code.

The development-system contains all components of the DBMS necessary for creating, maintaining, and executing applications, and must include the C-language interface. The development system is necessary for integrating the DBMS with BATMAN & ROBIN, writing the Object-Definition Database Editor, and adding new computer models to BATMAN & ROBIN. It should be purchased as a *site-license*, meaning any Sun workstation connected to the development-site local area network has access to the DBMS.

The end-user cost includes distribution royalties that would be incurred when delivering applications built with the DBMS. This cost must be considered because BATMAN & ROBIN is currently installed at a number of end-user sites and will be installed at many more in the future. Since all other BATMAN & ROBIN software can be distributed free to any DoD site, using a DBMS that requires an end-user license complicates installation and increases the cost to potential sites. For vendors that allow royalty-free distribution, this is not a problem. Therefore, a DBMS that has no distribution royalties is favored over one that requires an end-user license.

4.9 Future Enhancements

Since they are *future* enhancements to BATMAN & ROBIN, the enhancements listed at the end of section 2 are, by their nature, uncertain. In a sense, this is a flexibility criterion, used to ensure that a particular DBMS won't obstruct potential directions for BATMAN & ROBIN.

Candidate DBMS's are given a 1 if they would facilitate the enhancement and a -1 if they would obstruct implementation. The values assigned are by consensus of the development team. The following sections describe the criteria used to score each future enhancement.

4.9.1 Object-Definition Database Editor

To implement the Object-Definition Database Editor, an ideal DBMS would allow the programmer to:

- Add, delete, and modify database records, e.g., platforms in a platform database.
- Search a database using a key, e.g., search a platform database for all U.S. (the key) platforms.
- Search a database using multiple keys, e.g., search a platform database for all U.S. (first key) air (second key) platforms.
- Get the next record in a sorted list, e.g., retrieve the next record from a search of U.S. (first key), JTIDS-capable (second key), air (third key) platforms.

These features would benefit both the programmers and the users of the Object-Definition Database Editor. The programmers would be given a robust set of tools to work from; the users would see fast response time from efficient searches. Therefore, DBMS's that provide these features are given a point.

4.9.2 Adding Computer Models

Adding a computer model to BATMAN & ROBIN typically requires:

1. adding new fields to existing records, e.g., the antennas field to the platform record for JTIDS, and
2. creating new record types, e.g., sonobuoys for the ASW models.

Therefore, DBMS's that facilitate these types of changes are given a point.

4.9.3 Multiple-Machine Architecture

To date, the design of a multiple-machine architecture is uncertain. There are, however, two primary design strategies: a window server or a database server.

If multiple machines were coordinated using a window server, as in X-windows (Nye, 1988), a single server station would drive the displays of several client stations. In this case, the database would be managed by a single program on the server station. Under this implementation, a multiple-machine architecture would be transparent to the database manager. Hence, if the multiple-machine architecture is implemented as a window server, it has no effect on the DBMS recommendation.

However, if this enhancement is implemented using a database server, then it does have an effect on this report's recommendations. Therefore, as a conservative approach, this future enhancement is scored assuming that the multiple-machine architecture is implemented as a database server.

Under the database-server design, one copy of the database would reside on the server machine, and each client machine would access the database independently. To support this capability, a DBMS would need (1) record locking to provide multi-user contention arbitration and (2) network server support. Therefore, DBMS's that provide these two features are given a point.

4.10 Performance

It is difficult to obtain meaningful performance data for DBMS's. Some vendors publish VAX 11/

780 benchmarks for their packages; others offer abstract graphs illustrating the relative performance of their product against other products. Most provide no performance data at all, except to say that their products are faster than the competition's.

Ideally, the way to measure DBMS performance for BATMAN & ROBIN would be to purchase an evaluation copy of each, run it through a custom benchmark program on the Sun-4/260, and display the results in a figure or a table. This effort would be costly and time-consuming. Moreover, since most of the Object-Definition Database is read into memory-resident structures at BATMAN & ROBIN's startup, the runtime performance of candidate DBMS's is less critical than other criteria.

Therefore, because of the difficulties in objectively measuring performance, and because, to date, DBMS performance is not critical to BATMAN & ROBIN, this criterion is not included in the evaluation.

5.0 Evaluation

Table 2 presents an evaluation matrix of DBMS's. The products are listed vertically, while the criteria used to evaluate them are listed horizontally. The information gathered for each criterion is discussed in the following sections.

5.1 Software Interface

Currently, BATMAN & ROBIN's interface to the Object-Definition Database is through the two routines *d_get_i* and *d_get_s*. Occupying 75 lines of code, this represents the minimum amount of code that would have to be rewritten to integrate any of the commercial DBMS's or ndbm into BATMAN & ROBIN. This does not include the ".defaults file" conversion program that would have to be written (see section 4.5, **Defaults Conversion**). Therefore, the actual number of coding lines that would have to be written depends on the peculiarities of each DBMS, but for any DBMS the number would be greater than 75. Without doing the integration, though, it is impossible to get an accurate measure here.

If Sun's User Defaults Database were kept, no existing lines of code would have to be changed and no conversion program would have to be written.

Sybase, Gemstone, Ontos, C-tree, and db_VISTA III could be applied to BATMAN & ROBIN's Icon database since they support binary data.

5.2 C Compatibility

All DBMS's provide a C-language interface except for Ontos, which supports only C++.

5.3 Vendor Support

With the high number of commercial DBMS's available, it is not surprising that all vendors offer telephone technical support.

5.4 Product Stability

Most of the RDBMS's and DBMS toolkits meet BATMAN & ROBIN's stability requirements, as described in section 4.7, **Product Stability**. Sybase, Gemstone, Ontos, and C Database Toolkit,

however, do not meet the stability requirements.

Table 2

DBMS Evaluation

Product	Software[1] Interface	C	Support	Stability[2]	Cost[3]	Future[4] Scores
Informix	75+ / N	Y	Y	6 / thousands	$2,050 / $925	1 / -1 / 1
Ingres	75+ / N	Y	Y	9 / thousands	$15,160 / $4,000	1 / -1 / 1
Oracle	75+ / N	Y	Y	8 / thousands	$11,780 / $5,130	1 / -1 / 1
Sybase	75+ / Y	Y	Y	3 / 200+	$10,800 / $6,480	1 / -1 / 1
Gemstone	75+ / Y	Y	Y	2.5 / 90	$32,000+ / Ng[5]	1 / -1 / 1
Ontos	75+ / Y	N	Y	0.75 / 80	$15,000 / Ng[5]	1 / -1 / 1
CDB Toolkit	75+ / N	Y	Y	8 / 150	$295 / $150	1 / -1 / 1
C-ISAM	75+ / N	Y	Y	10 / thousands	$630 / $315	1 / -1 / 1
C-tree	75+ / Y	Y	Y	10 / 10,000+	$395 / $0	1 / -1 / 1
db_VISTA	75+ / Y	Y	Y	6 / 8,000+	$4,490 / $0	1 / -1 / 1
ndbm	75+ / N	Y	Y	3+ / 250,000+	$0 / $0	1 / -1 / -1
Sun User DB	0 / N	Y	Y	3+ / 250,000+	$0 / $0	-1 / 1 / -1

[1]Number of coding lines / can be applied to BATMAN & ROBIN's Icon Database.

[2]Number of years on market / number of units sold.

[3]Development-system / end-user .

[4]Object-Definition Database Editor / Adding Computer Models / Multiple Machine Architecture. For more information, refer to section 5.6, **Future-Enhancement Scores.**

[5]Negotiable, but present.

5.5 Cost

Only ndbm and Sun User DB are *totally* free to BATMAN & ROBIN. C-tree and db_VISTA are free to end-user sites, but require a development-system cost that would be incurred by NPRDC and transition sites. All other DBMS's have development- and end-user costs.

The cost quotes for each DBMS are broken down as follows.

Informix: The data listed refers to the two Informix products *INFORMIX-SQL*, providing an SQL interface to the database, and *INFORMIX ESQL/C*, providing the C-language interface. The development-system cost is $2,050. The end-user cost is $925.

Ingres: The development-system cost includes (1) $13,200 for two to eight users, and (2)

$1,960 for the embedded SQL C-language interface. The end-user cost is $4,000.

Oracle: The development-system cost includes (1) a database-engine cost of $10,450 for two to 16 users, (2) $760 for the SQL interface (SQL*Plus), and (3) $570 for the C-language interface (PRO*C). The end-user cost includes (1) a database-engine cost of $3,800, (2) $760 for SQL*Plus, and (3) $570 for PRO*C.

Sybase: The development-system cost includes (1) $10,000 for two to four users, and (2) $800 for the C-programmer's interface. The end-user cost includes (1) $6,000 for the base system and (2) $480 for the C-programmer's interface.

Gemstone: The development-system cost of $32,000 is for four users. The end-user cost is negotiable, depending on the number of units sold.

Ontos: The development-system cost is $15,000. The end-user cost is negotiable, depending on the number of units sold.

C Database Toolkit: The development-system cost of $295 is for the CDB library, the C-language interface. The end-user cost is $150.

C-ISAM: The development-system cost is $630. The end-user cost is $315.

C-tree: The development-system cost of $395 includes source code. There is no end-user cost.

db_VISTA III: The development-system cost of $4,490 includes source code. There is no end-user cost.

ndbm: Since it comes standard with SunOS, there is no development-system or end-user cost.

Sun User DB: Since it comes standard with SunOS, there is no development-system or end-user cost.

5.6 Future-Enhancement Scores

The following sections outline how the future-enhancement scores for each DBMS were determined. As described in section 4.9, **Future Enhancements**, a 1 means the DBMS would facilitate the enhancement and a -1 means it would obstruct implementation.

5.6.1 Object-Definition Database Editor

The commercial DBMS's and ndbm all provide capabilities that would ease implementing the Object-Definition Database Editor. Therefore, these DBMS's were given a 1 for this future enhancement.

Sun's User Defaults Database, however, is not designed to provide these capabilities, and was given a -1. The limitations of Sun's User Defaults Database are discussed in section 2.0, **Introduction**.

5.6.2 Adding Computer Models

For the commercial DBMS's and ndbm, performing the tasks required to add a computer model can be difficult. At the minimum, the programmer modifies the record structure and then runs an update program included with the DBMS. In the worst case, the update program has to be written by the programmer from scratch. Because of this added overhead, the commercial DBMS's and ndbm were all given a -1 for this future enhancement.

Sun's User Defaults Database, however, is easy for programmers to update since it is stored as a standard text file in ASCII format. The programmer can use vi or some other UNIX text editor to make the additions. Therefore, Sun's User Defaults Database was given a 1 for this future enhancement.

5.6.3 Multiple-Machine Architecture

All DBMS's except ndbm and Sun's User Defaults Database provide record locking and network server support. Therefore, only these two were given a -1 for this future enhancement.

6.0 Conclusions

The RDBMS's discussed in this report (Informix, Ingres, Oracle, and Sybase) are robust, rating favorably in many categories. The only notable exception is Sybase's limited time on the market and small installed base. In all other regards, these are capable products that could benefit BATMAN & ROBIN. However, when considering the issues surrounding BATMAN & ROBIN, there are a number of problems with these RDBMS's. First, they are expensive, both in development-system and end-user costs. Second, they provide far more functionality than BATMAN & ROBIN needs, for example, SQL and 4GL. This can complicate integrating these systems into BATMAN & ROBIN, since there would be a learning curve attached to these products. Lastly, one of the main advantages of RDBMS's, their comprehensive set of utilities, is not relevant to BATMAN & ROBIN since the Object-Definition Database Editor will be developed. That is, BATMAN & ROBIN would not use the form, menu, and report-generating utilities that often come standard with RDBMS's. If an RDBMS is chosen, Informix seems the best because, while it rates equal to the other products in other regards, it is less expensive.

OO-DBMS's have two principal advantages: (1) they can store binary data, e.g., icons and (2) their designs are consistent with BATMAN & ROBIN. However, these systems have a number of disadvantages that far outweigh these advantages. First, they are very expensive, even more so than RDBMS's. This high cost is due in part to the second disadvantage of OO-DBMS's, their small installed base. Lastly, similar to RDBMS's, integration with these systems would be difficult because of the required learning curve. Addressing the two OO-DBMS's discussed in this report, Gemstone rates better than Ontos, since Gemstone is more stable and since Ontos does not have a C-language interface.

DBMS toolkits provide many of the capabilities needed by BATMAN & ROBIN, but are much less expensive than RDBMS's and OO-DBMS's. Four important points to note concerning the toolkits are (1) both C Database Toolkit and C-ISAM require end-user distribution royalties; (2) C Database Toolkit does not pass the product-stability requirements since it has sold only 150 units; (3) db_VISTA III's development system cost is much higher than the others; and (4) both ndbm and User Defaults Database, since they come bundled with SunOS, have no development-system

or end-user cost.

7.0 Recommendations

The strongest selection criteria discovered in this report are (1) C compatibility and (2) cost (in its varying forms). As illustrated in the decision tree in Figure 1, these two criteria are the basis for the recommendations that follow.

Since BATMAN & ROBIN is written in the C programming language, any candidate DBMS must provide a C-language interface. Ontos does not, and this report recommends against it.

This report also recommends avoiding any DBMS that requires end users to pay distribution royalties. Since BATMAN & ROBIN can currently be distributed without cost to any DoD site, adding a distribution royalty would increase the expense and logistics of delivering BATMAN & ROBIN and would reduce the system's appeal. Furthermore, with the large number of potential end-user sites throughout the DoD, the amount paid in distribution royalties could be substantial. Therefore, this report recommends against Informix, Ingres, Oracle, Sybase, Gemstone, Ontos, C Database Toolkit, and C-ISAM.

An additional reason for avoiding the RDBMS's and OO-DBMS's is their high development-system cost. The DBMS toolkits provide sufficient functionality for BATMAN & ROBIN but at a much lower price.

After applying the C-language filter and the distribution-royalties filter, the remaining DBMS's are: C-tree, db_VISTA III, ndbm, and User Defaults Database. The following summary lists the advantages and disadvantages of each.

C-tree* and *db_VISTA III

Advantages:

- Are very stable products.
- Have no end-user distribution royalties.
- Can store icons using variable length records.
- Could be used in a multiple-machine architecture since they both provide record locking and network server capabilities.
- Source code is included.
- Contain a robust set of programming tools.

Disadvantages:

- Transition sites must purchase a development-system.
- Difficult for programmer to add new fields and records.

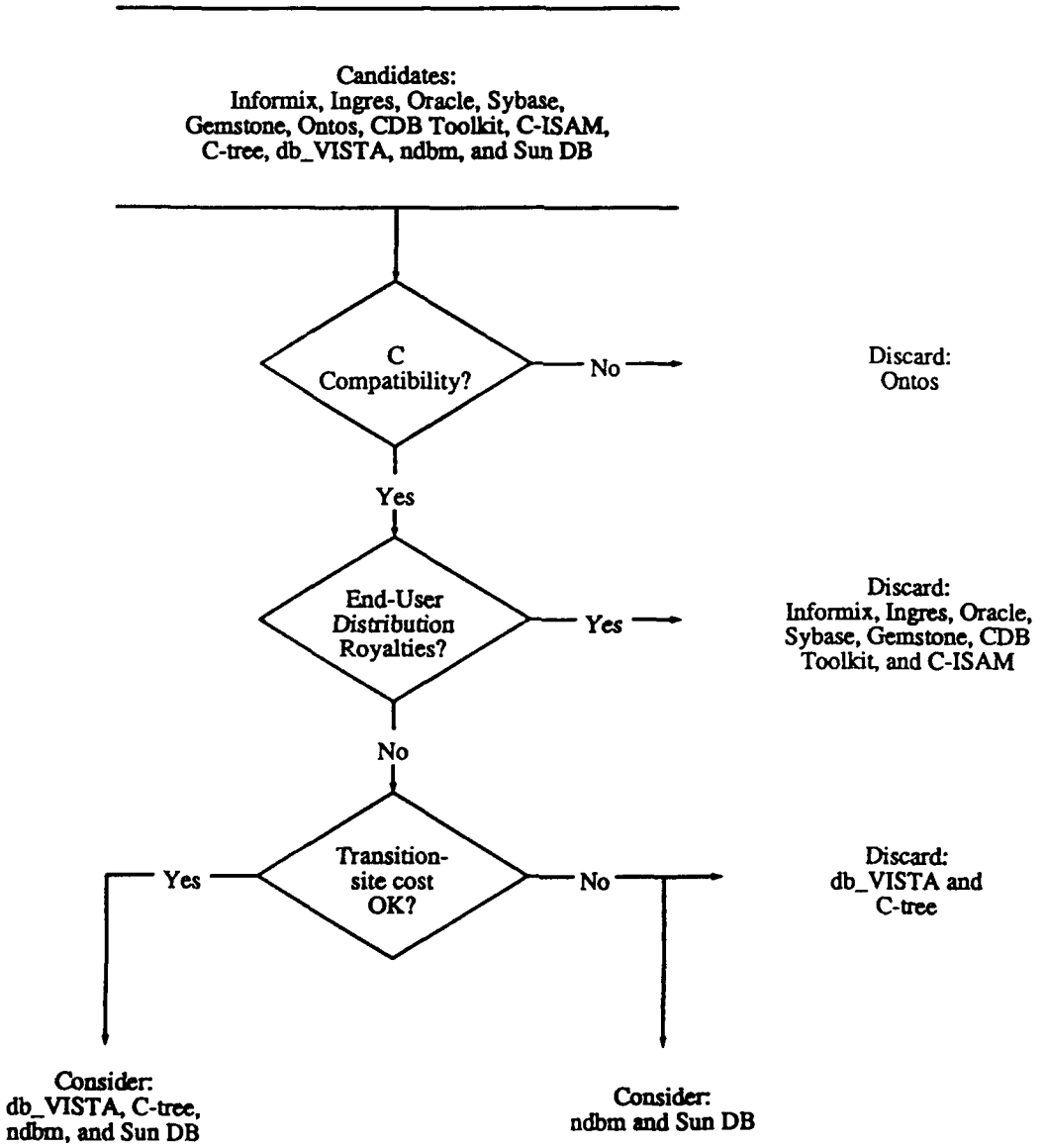


Figure 1. DBMS-Selection Decision Tree

ndbm

Advantages:

- No development or end-user cost.
- Contains a satisfactory set of programming tools.

Disadvantages:

- Difficult for programmer to add new fields and records.
- Cannot store icons.
- Suffers in a multiple-machine architecture because it does not provide record locking or network server capabilities.
- Source code is not available.

User Defaults Database

Advantages:

- No development or end-user cost.
- Easy for programmer to add new fields and records.

Disadvantages:

- Obstructs developers from using this database for its intended purpose.
- Cannot store icons.
- Suffers in a multiple-machine architecture because it does not provide record locking or network server capabilities.
- Source code is not available.
- Contains a limited set of programming tools.

As a final cost-related selection filter, note that transition sites that use BATMAN & ROBIN as a point of departure must purchase a development-system version of the DBMS. If this cost is not acceptable, then C-tree and db_VISTA III should not be considered because of their development-system costs. Under this condition, this report recommends experimenting with ndbm to discover firsthand how it might benefit BATMAN & ROBIN. Perhaps both ndbm and Sun's User Defaults Database could be used to implement the Object-Definition Database. That way, the advantages of one system might be used to counterbalance the disadvantages of the other.

However, if transition-site cost is acceptable, or if there is such a small number of transition-sites that it is not an issue, the potential benefits of C-tree and db_VISTA III should not be overlooked.

In particular, both C-tree and db_VISTA III would be useful for developing the Object-Definition Database editor and for implementing a multiple-machine architecture, while ndbm and User Defaults Database may complicate these enhancements. Therefore, under the condition that transition-site cost is acceptable, this report recommends experimenting with C-tree and db_VISTA III, as well ndbm, to get a better idea of the capabilities of each system. Both C-tree and db_VISTA III offer trail evaluation copies of their products. Again, Sun's User Defaults Database could be used in conjunction with the selected DBMS to reduce the disadvantages of either system.

References

Butler, C.W., Hodil, E.D., and Richardson, G.L. "Building Knowledge-Based Systems with Procedural Languages." IEEE Expert, Summer 1988, pp. 47-59.

Federico, P-A., Bickel, Ullrich, Bridges, and Van De Wetering. BATMAN & ROBIN Rationale, Software Design, and Database Descriptions (TN 89-18). Navy Personnel Research and Development Center. San Diego, CA: April 1989.

Gevarter, William B. The Nature and Evaluation of Commercial Expert System Building Tools. NASA Ames Research Center. Moffett Field, CA: May 1987.

Kernighan, B.W., and Ritchie, D.M. The C Programming Language. Second Edition. Englewood Cliffs NJ: Prentice-Hall, 1988.

Meyer, Bertrand. Object-Oriented Software Construction. Hemel Hempstead: Prentice-Hall International, 1988.

Nye, Adrian. XLIB Programming Manual. Newton, MA: O'Reilly & Associates, INC., 1988.

Peterson, Robert W. "Object-Oriented Data." AI Expert, March 1987, pp. 180-185.

Programmer's Shop. Catalog, Spring 1990, p. 41.

Rodgers, Ulrika. UNIX Database Management Systems. Englewood Cliffs, NJ: Prentice-Hall, 1990.

Sun Microsystems. SunOS Reference Manual. Mountain View CA: 1990.

Sun Microsystems. SunView System Programmer's Guide. Mountain View CA: 1990.

Sun Microsystems. Catalyst Catalog. SPARC Edition. Mountain View CA: Fall 1989.

Tucker, Michael Jay. "Object-Oriented Databases Arrive." UNIXWORLD, August 1989, pp. 62-66.

Wirth, Niklaus. Algorithms + Data Structures = Programs. Englewood Cliffs NJ: Prentice-Hall, 1976.

Databases and Vendors

C Database Toolkit	Jaybe Software 2509 N. Campbell, Suite 259 Tuscon, AZ 85719 (602) 327-2299
C-ISAM, Informix	Informix Software, Inc. 4100 Bohannon Drive Menlo Park, CA 94025 (415) 322-4100
C-tree	Faircom 4006 West Broadway Columbia, MO 65203 (314) 445-6833
db_VISTA III	Raima Corporation 3245 146th Place S.E. Bellevue, WA 98007 (206) 747-5570
Ingres	Relational Technology, Inc. 1080 Marina Village Parkway P.O. Box 4006 Alameda, CA 94501 (415) 769-1400
Gemstone	Servio Logic Corporation 1420 Harbor Bay Parkway Suite 100 Alameda, CA 94501 (415) 748-6200
ndbm, User Defaults Database	Sun Microsystems, Inc. 2550 Garcia Avenue Mountain View, CA. 94043 (415) 960-1300
Ontos	Ontologic Corporation Three Burlington Woods Burlington, MA 01803 (617) 272-7110
Oracle	Oracle Corporation 20 Davis Drive Belmont, CA 94002 (415) 598-8000

Sybase

Sybase, Inc.
6475 Christie Avenue
Emeryville, CA 94608
(415) 596-3500

DISTRIBUTION LIST

Office of the Chief of Naval Research (OCNR-20), (OCNR-222), (OCNR-1142), (OCNR-1142CS)
Bureau of Naval Personnel (PERS-01JJ), (PERS-11EE)
Space and Naval Warfare Systems Command (SPAWAR-159-4)
Chief of Naval Operations (OP-593), (OP-593D)
Naval Air Systems Command (PMA-205), (933G)
Commander, Naval Air Force, U.S. Pacific Fleet (313)
Naval Air Development Center (40L)
Naval War College (331)
Naval Research Laboratory (5530)
Naval Surface Warfare Center, Dahlgren (N31)
Naval Surface Warfare Center, Silver Spring (D25)
Naval Training Systems Center (254)
Naval Warfare Analysis Center (30M2)
Naval Weapons Center (3032)
Naval Postgraduate School (OR/Pp), (OR/Wg), (OR/B1), (OR/Na), (OR/Ha), (OR/Er)
Naval Ocean Systems Center (432), (44), (442), (444), (624), (713), (723)
Fleet Combat Training Center, Pacific (32)
Tactical Training Group, Pacific (N1)
Carrier Airborne Early Warning Weapons School (CO)
Johns Hopkins University Applied Physics Laboratory (WAL)
Defense Advanced Research Projects Agency (DIRO), (ASTO)
Army Research Institute ((PERI-ZT)
Air Force Human Resources Laboratory (OT), (LRS-TDC)
Institute for Defense Analyses, Science and Technology Division
Canadian Defense and Civil Institute of Environmental Medicine
Defense Technical Information Center (DTIC 2)